A Mobile Service Robot for Automisation of Sample Taking and Sample Management in a Biotechnological Pilot Laboratory

Dipl.-Inform. Torsten Scherer

Dept. of Cell Culture Technology,
University of Bielefeld
Dept. of Robotics and Embedded Systems,
Technical University of Munich

October 2004

A thesis submitted to the Faculty of Technology of the
University of Bielefeld in partial fulfilment
of the requirements for the degree

„Doktor der Ingenieurswissenschaften (Dr.-Ing.)“

Printed on ageing-resistant paper / Gedruckt auf alterungsbeständigem Papier (DIN-ISO 9706).

Dipl.-Inform. Torsten Scherer

AG Cell Culture Technology
Faculty of Technology
University of Bielefeld

AG Robotics and Embedded Systems
Institute of Informatics
Technical University of Munich

A thesis submitted to the Faculty of Technology of the University of Bielefeld in partial fulfilment of the requirements for the degree „Doktor der Ingenieurswissenschaften (Dr.-Ing.)“.

Printed on ageing-resistant paper (DIN-ISO 9706).

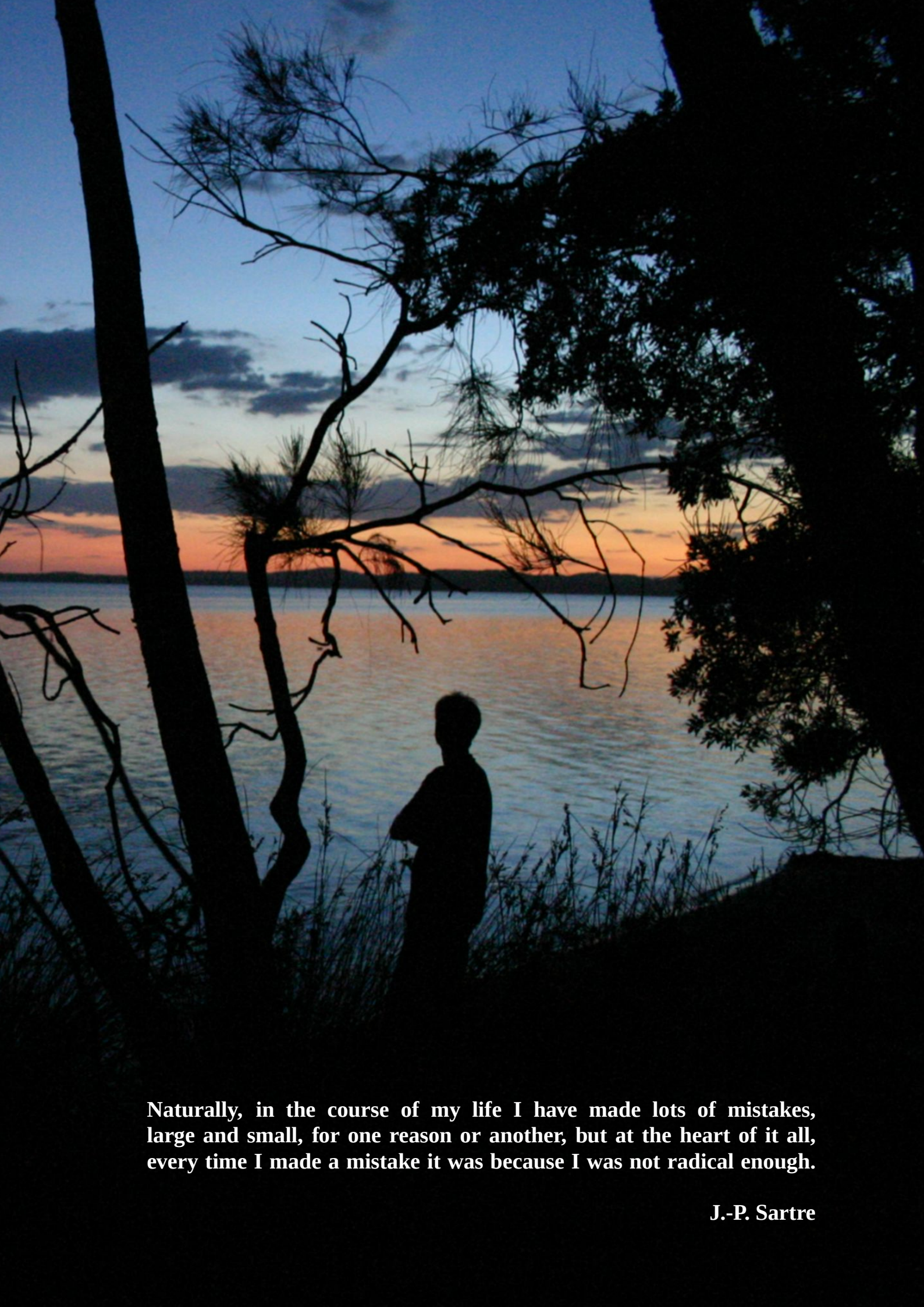
submitted: **27.10.2004**
defended and approved: **13.04.2005**

Reviewers:

Prof. Peter B. Ladkin, Ph.D.
Prof. Dr. Jianwei Zhang
Prof. Dr.-Ing. Alois C. Knoll

Members of the board of examiners:

Prof. Dr.-Ing. Jürgen Lehmann
Prof. Peter B. Ladkin, Ph.D.
Prof. Dr. Jianwei Zhang
Dr. Heino Büntemeyer



Naturally, in the course of my life I have made lots of mistakes, large and small, for one reason or another, but at the heart of it all, every time I made a mistake it was because I was not radical enough.

J.-P. Sartre

Acknowledgements

This work has mainly been financed by the “Arbeitsgemeinschaft industrieller Forschung” (AiF[1]) “Otto von Guericke e.V.” with funds from the German Federal Ministry of Economics and Labour (BMWA[2]) as projects AiF Nr. 11736 N/1 and AiF Nr. 13223 N/1+2. As for that, I’d like to thank the german taxpayer for the sponsoring, and the companies B. Braun Biotech International (Melsungen, Germany), Diessel (Hildesheim, Germany) and Innovatis AG (Bielefeld, Germany) for supporting the project.

It has also been substantially sponsored by Bayer Healthcare, Berkeley (USA), where I’d like to thank Dr. Konstantin Konstantinov and Dr. Rüdiger Heidemann for their support and interest in this new approach of automating biotechnological processes.

Concerning the non-financial support, I’d primarily like to thank my reviewers Professor Peter Ladkin, Professor Jianwei Zhang and Professor Alois Knoll for picking up the trouble of reading this pile of leaved tree produce and writing the reviews – I know this thesis is somewhat longer than usual, but that’s only because the project did cover more aspects than usual. This is especially valid for Prof. Ladkin, whom I know since he came to Bielefeld as a *“British citizen and (...) US Permanent Resident Alien”* as his bio claims, because he kindly agreed into writing a review without being directly involved in the project.

In addition to them I’d like to thank Professor Alois Knoll and Professor Jürgen Lehmann for their canoe trip during which they first came up with the idea for the project – it can’t have been a whitewater rafting trip if they’ve had spare time for thoughts like these. In the years 1999 to 2004 I’ve been working on this project, first in Prof. Knoll’s group in Bielefeld together with Dr. Zhang, and later – after Prof. Knoll moved to the Technical University of Munich and Dr. Zhang moved to the University of Hamburg to become Professor Zhang – technically/financially in Prof. Knoll’s group in Munich, but physically in Prof. Lehmann’s group in Bielefeld together with Dr. Dirk Lütkemeyer. This interdisciplinary experience has been very interesting and I owe all the people a lot.

The interdisciplinary character of this work means that I’ve had contact with a lot of members of two groups, a computer science group and a biotechnological group. On the computer science side I’d like to thank my former colleagues or fellow computer/robot freaks Tim Baier, Yorck von Collani, Markus Ferch, Axel Schneider, Hagen Stanek and Daniel Westhoff for contributing to this work by means of ideas, suggestions, encouragements, criticisms and corrections, as well as all the other members of the old AG-TI for being a great group: Ingo Glöckner, Thorsten Graf, Ingo Homann, Dirk Müller (*öpening beenary mood däda gonnectium* :-), Frank Röben and Christian Scheering. Life has not always been easy in the eventful times when the old group was re-organised, and I am not entirely sure whether I would have finished this work without their support.

[1] `http://www.aif.de`
[2] `http://www.bmwi.de`

On the biotechnological side I'd primarily like to thank my colleague Iris Poggendorf for being the complementary part of the project – without her the robot would stand silently and have nothing to do. Also important for me and the project were Heino Büntemeyer and his technical and administrative support, Dirk Lütkemeyer as the general organiser of things and Hans Brinkmann as the general fixer of things. This shall by no means exclude any other member of the group, but they are just too many – may they forgive me.

Finally, I would like to thank Douglas N. Adams for providing the holistic introductory words for each chapter and the wise insights into the very foundations of human life – may he rest in peace[3].

[3] See [Adams 1979]. `http://www.douglasadams.com`

Publications

Excerpts of this work have been published in/on the

- Prozessmesstechnik in der Biotechnologie 2000, Tagung, Bamberg, Germany
  [Scherer et. al. 2000]
- Cell Culture Engineering VII, Santa Fe, USA
  [Lütkemeyer et. al. 2000a]
- 11th International Biotechnology Symposium, Berlin, Germany
  [Lütkemeyer et. al. 2000b]
- Biotechnol. Prog. 16, No. 5, 2000
  [Lütkemeyer et. al. 2000c]
- 17th Esact-Meeting, Tylosand, Sweden
  [Lütkemeyer et. al. 2001]
- 8th Int. Conference on Computer Applications in Biotechnology, Québec City, Canada
  [Poggendorf et. al. 2001]
- BioTec - Fachmagazin für Biotechnologie, Nr. 9-10, Vereinigte Fachverlage, Germany
  [Poggendorf and Scherer 2002]
- GVC/Dechema Jahrestagungen 2002, Wiesbaden, Germany
  [Poggendorf et. al. 2002a]
- Cell Culture Engineering VIII, Snowmass, USA
  [Poggendorf et. al. 2002b]
- 6. Dresdner Sensor-Symposium, Dresden, Germany
  [Poggendorf et. al. 2003]
- IEEE Conf. on Emerging Technologies and Factory Automation 2003, Lisbon, Portugal
  [Scherer et. al. 2003]
- Cell Culture Engineering IX, Cancun, Mexico
  [Poggendorf et. al. 2004]
- Robotik 2004 Conference, München, Germany
  [Westhoff et. al. 2004a]
- Automatisierungstechnische Praxis 10/2004, Oldenbourg Industrieverlag, Deutschland
  [Westhoff et. al. 2004b]

Contents

I Thesis **1**

1 Introduction **3**
1.1 Motivation . . . 3
1.2 State of the Art . . . 4
1.3 About this Work . . . 10
1.4 Thesis Structure . . . 11
1.5 Terminology . . . 12
1.6 A Note on HTTP Links . . . 13

2 Automating a Biotechnological Laboratory **15**
2.1 Biotechnology . . . 16
2.1.1 Cells Compounds versus Single Cells . . . 16
2.1.2 Cultivation of Mammalian Cells in Pilot Scale . . . 16
2.1.3 Sample Management . . . 17
2.1.4 Preliminary Task Definition . . . 20
2.2 The Automation System Architecture . . . 20
2.2.1 The Laboratory . . . 22
2.2.2 The Sampling Device(s) . . . 23
2.2.3 The Pipetting Device . . . 24
2.2.4 The Cell Counter . . . 25
2.2.5 The Centrifuge . . . 26
2.2.6 The Fridge . . . 27
2.3 Task Definition . . . 28
2.4 Robot Architecture . . . 30

3 Accurate Positioning of a Mobile Platform **33**
3.1 The “Neobotix MP-L655“ Mobile Platform . . . 36
3.2 Localisation . . . 38
3.2.1 Sensors . . . 39
3.2.1.1 Odometry . . . 40
3.2.1.2 Gyro Compass . . . 41
3.2.1.3 Laser Scanners . . . 42
3.2.2 Kinematics Issues . . . 45
3.2.3 Dead Reckoning . . . 46
3.2.3.1 Improving the Kinematic Model . . . 47
3.2.3.2 Borenstein Tests . . . 50
3.2.3.3 Summary . . . 52
3.2.4 Least Squares Fit . . . 52
3.2.4.1 Initial Guess . . . 52

3.2.4.2 Least Squares Fit . . . 53
3.2.4.3 Summary . . . 55
3.2.5 Kalman Filter . . . 55
3.2.5.1 System and Measurement Model . . . 56
3.2.5.2 Predict and Update . . . 58
3.2.5.3 Summary . . . 59
3.2.6 Extended Kalman Filter . . . 60
3.2.6.1 Simplifications . . . 60
3.2.6.2 Summary . . . 62
3.2.7 The MP-L655 Extended Kalman Filter . . . 62
3.2.7.1 State Definitions . . . 63
3.2.7.2 Models . . . 64
3.2.7.3 Model without Acceleration . . . 66
3.2.8 Kalman Filtering Summary . . . 68
3.2.8.1 General Issues . . . 68
3.2.8.2 Localisation Issues . . . 71
3.2.8.3 Numerical Issues . . . 72
3.2.8.4 Consequences . . . 73
3.2.9 Realisation . . . 73
3.3 Path-Planning . . . 74
3.3.1 Maps . . . 74
3.3.1.1 Map Design . . . 75
3.3.1.2 Map Expansion . . . 76
3.3.2 Graphs . . . 82
3.3.2.1 Visibility Graph . . . 82
3.3.2.2 Tangent Graph . . . 84
3.3.3 Path Search . . . 85
3.3.3.1 Arbitrary Start and Goal Points . . . 85
3.3.3.2 A^* Algorithm . . . 86
3.3.4 Summary . . . 87
3.4 Motion Execution . . . 89
3.4.1 Kinematics and Motion Types . . . 89
3.4.1.1 Neobotix Software . . . 90
3.4.1.2 Tracking & Inverse Kinematics . . . 91
3.4.1.3 Conclusions . . . 91
3.4.2 Trajectory Generation Filter with Velocity Profiles . . . 92
3.4.2.1 Velocity Profiles . . . 92
3.4.2.2 Position Setpoints . . . 95
3.4.2.3 Velocity Setpoints . . . 96
3.4.2.4 Summary . . . 97
3.4.3 Absolute Rotations . . . 98
3.4.4 Absolute Translations . . . 98
3.4.5 Relative Translations . . . 100
3.4.6 Collision Avoidance . . . 101
3.4.7 Realization & Enhancements . . . 102
3.4.8 Summary . . . 104
3.5 Future Work . . . 104
3.5.1 Sensors . . . 104

3.5.2 Localisation . . . 106
3.5.3 Path-Planning . . . 109
3.5.4 Behaviours . . . 110
3.6 Experimental Results . . . 111
3.6.1 Localisation Position . . . 113
3.6.2 Absolute Verification of Localisation Position . . . 116
3.6.3 Relative Visual Verification of Localisation Position . . . 118
3.6.4 Summary . . . 121

4 Advanced Robot Arm Control **131**
4.1 The Robot Arm & Tool . . . 133
4.1.1 The “Mitsubishi PA10” Robot Arm . . . 133
4.1.2 The Tool . . . 134
4.2 Robot Control Software . . . 136
4.2.1 The PA-Library . . . 136
4.2.2 The “Robot Control C-Library” RCCL . . . 137
4.2.3 Consequences . . . 139
4.3 RCCL on the PA10 . . . 139
4.3.1 Kinematic . . . 140
4.3.2 ARCNet Interface and Protocol . . . 142
4.3.2.1 PA10 ARCNet Hardware . . . 143
4.3.2.2 PA10 Protocol . . . 144
4.3.3 Joint Controllers . . . 145
4.4 Results . . . 147
4.4.1 Path Tracking Performance . . . 148
4.4.1.1 Cartesian spiral Motion . . . 148
4.4.1.2 Cartesian square Motion . . . 151
4.4.2 Force Control . . . 152
4.4.3 Summary . . . 155

5 Robust Colour Vision **159**
5.1 The Implemented Colour Vision System . . . 160
5.1.1 Why Colour? . . . 160
5.1.2 Vision Hardware . . . 162
5.1.3 System Structurisation . . . 164
5.2 Image Segmentation . . . 165
5.2.1 Smart Smoothing . . . 165
5.2.2 Colour Similarity . . . 167
5.2.2.1 Fixed Colours . . . 172
5.2.2.2 Adaptive Colours . . . 174
5.2.3 Flood Fill . . . 176
5.3 Object Recognition & Displacement Computation . . . 177
5.3.1 Object Models . . . 181
5.3.2 Model Generation . . . 183
5.3.3 Model Matching . . . 184
5.3.3.1 Displacement Model . . . 184
5.3.3.2 Least-Squares Fit . . . 185
5.3.3.3 Permutations . . . 188
5.4 Iterative Displacement Compensation . . . 189

5.5 Experiments & Results . . . 190
5.5.1 Experiments . . . 191
5.5.1.1 Basic Repeat Accuracy . . . 192
5.5.1.2 Robot Arm Positioning Noise . . . 194
5.5.1.3 Influence of Vision Parameters . . . 197
5.5.1.4 Planar Marker Displacement . . . 200
5.5.1.5 Tilted Marker Displacement . . . 202
5.5.1.6 Centrifuge Displacement . . . 202
5.5.1.7 Classification Error Comparison . . . 205
5.5.2 Summary . . . 206
5.5.3 Future Work & Applications . . . 208

6 Realisation and Integration 211
6.1 Robot Command Scripts . . . 212
6.1.1 Custom Script Language . . . 213
6.1.2 Script Language Commands . . . 214
6.1.3 Example . . . 215
6.2 Robot State Automaton . . . 218
6.3 Database . . . 224
6.3.1 Database Considerations . . . 224
6.3.2 INFOBASE . . . 225
6.4 Networking . . . 226
6.4.1 TCP/IP Networking . . . 226
6.4.2 CNETOBJ . . . 228
6.5 Network Support for Laboratory Devices . . . 228
6.5.1 CLABMSG / CLABDEV / CLABDEVD . . . 229
6.5.2 Laboratory Devices . . . 230
6.6 The Networked Robot . . . 231
6.6.1 CROBOT / CROBOTD . . . 231
6.6.2 robotd . . . 233
6.6.3 Modifying/Extending the Functionality . . . 234
6.7 Summary . . . 235

7 Automated Sample Management Results 239
7.1 Automated Sample Management . . . 239
7.1.1 The Sample Management Sequence . . . 241
7.1.2 The Robot Actions . . . 245
7.2 Biotechnological Results . . . 255
7.3 Summary . . . 256

8 Conclusions & Future Work 259
8.1 The Robot System . . . 260
8.2 Roblets . . . 264
8.3 Applications . . . 265

Bibliography 267

Index 276

II Appendices **281**

A Mobile Platform Documentation **283**
A.1 The “Neobotix MP-L655“ Mobile Platform . . . 283
A.1.1 Hardware . . . 283
A.1.2 Software . . . 286
A.2 The CAN Interface . . . 289
A.2.1 CAN Hardware . . . 289
A.2.2 CAN Frames . . . 289
A.2.3 The C164 Motor Controller(s) . . . 291
A.2.4 The C167 I/O Controller . . . 294
A.3 The LMS-200 Serial Communication . . . 296
A.3.1 Serial Communication Hardware Issues . . . 296
A.3.2 Serial Communication Software Issues . . . 297
A.3.3 Telegrams and Modes . . . 298
A.3.4 Summary . . . 299
A.4 mobiled . . . 300

B Robot Arm Documentation **303**
B.1 The “Mitsubishi PA10” Robot Arm . . . 303
B.1.1 Level 1 – The PA10 Arm . . . 303
B.1.2 Level 2 – The PA10 Controller . . . 305
B.1.3 Level 3 – The PA10 Motion Control Board / PA-Library . . . 306
B.1.4 Newer PA10 Versions . . . 307
B.2 The Robot Tool . . . 309
B.2.1 The “ATI Force Torque Sensor” . . . 309
B.2.2 The “phd Electrical Gripper” . . . 311
B.2.3 Gripper Fingers . . . 312
B.3 RCCL . . . 314
B.3.1 Mathematic Computations . . . 314
B.3.1.1 Transforms . . . 314
B.3.1.2 Kinematic . . . 315
B.3.2 Position Equations . . . 316
B.3.3 Arbitrary Motions . . . 318
B.3.4 Singularities . . . 320
B.3.5 (Virtual) Robot Cooperation . . . 321
B.3.5.1 Cooperation . . . 321
B.3.5.2 Virtual Robots . . . 322

C Vision System Documentation **325**
C.1 Vision Hardware . . . 325
C.1.1 JAI M1250 Microhead Colour Camera . . . 325
C.1.2 Matrox Meteor Framegrabber . . . 327
C.2 Introduction to Computer Vision . . . 328
C.2.1 Image Representation . . . 328
C.2.2 Colour Spaces . . . 330
C.2.2.1 RGB . . . 330
C.2.2.2 YUV . . . 333
C.2.2.3 HSV / HSI / HSL . . . 335

C.2.3 Linear Image Filtering . . . 336
C.2.3.1 Convolutions . . . 337
C.2.3.2 Smoothing . . . 338
C.2.3.3 Laplace Filter . . . 339
C.2.3.4 Sobel Filter . . . 339
C.2.4 Non-Linear Image Filtering . . . 340
C.2.4.1 Median . . . 341
C.2.4.2 Smart Smoothing . . . 341
C.2.4.3 SUSAN . . . 342
C.2.5 Segmentation . . . 343
C.2.5.1 Top-Down Approaches . . . 343
C.2.5.2 Bottom-Up Approaches . . . 345
C.2.6 Feature Based Classification . . . 345
C.2.6.1 Neural Networks . . . 347
C.2.6.2 Fuzzy Controllers . . . 349
C.2.7 Appearance Based Classification . . . 350
C.2.7.1 Principal Component Analysis . . . 351
C.2.7.2 Output Related Features . . . 354
C.2.8 Summary . . . 355

D C++ Class Documentation 357
D.1 CINFOBASE – Accessing the INFOBASE . . . 357
D.2 CNETOBJ – Communicating over a TCP/IP network . . . 361
D.2.1 CCHUNK . . . 361
D.2.2 CNETOBJ . . . 365
D.3 CLABDEV – Control of Laboratory Devices . . . 367
D.3.1 CLABMSG . . . 367
D.3.2 CLABDEV . . . 368
D.3.3 CLABDEVD . . . 368
D.4 CROBOT – Accessing the Mobile Robot System . . . 369
D.4.1 CROBOT . . . 369
D.4.2 CROBOTD . . . 373
D.5 CMOBILE . . . 379

E Nonsense 383

Part I

Thesis

Chapter 1

Introduction

"Unfortunately I got stuck on the earth for rather longer than I intended."

Ford Prefect

1.1 Motivation

The thesis presented here is part of the results of a collaborative and interdisciplinary project between the groups of *Cell Culture Technology*[1] of Prof. Dr.-Ing. J. Lehmann and *Technical Computer Science*[2] of Prof. Dr.-Ing. A. Knoll[3] at the University of Bielefeld, Germany. The project with the original (german) title of

> „Roboterisierung der Probenentnahme und des Probenmanagements bei der Kultivierung tierischer Zellen im Pilotmaßstab[4]"

involves two of todays key high-tech research topics: biotechnology and robotics. Biotechnology, on the one hand side, is a modern field of research, but many of its processes are still not automated. This is particularly true for pilot scale plants where new processes are developed. Instead, these processes require manual labour by human personnel, which with the never-ending discussion about wages and non-wage labour costs is increasingly becoming a problem. Robotics, on the other hand side, is a well established field of research designing machines to help humans at repetitive and tedious tasks. The probably most commonly known example is the car industry, where robots are being used to assemble cars or car parts for a very long time. These systems are, however, typically very inflexible.

The goal of the project is to bring together these two fields of research in a feasibility study trying to automate the sample management – a complex and particularly tedious part of a biotechnological process involving numerous spatially separated devices. Its tediousness arises from the fact that – for mammalian cells – it has to be done every couple of hours for a period of up to several weeks, which explicitly includes nights and weekends. No employee likes to work at nights and weekends and no employer likes to pay the employees to do so, so an automated solution would be highly welcome by everyone.

[1] AG Zellkulturtechnik, Technische Fakultät, Universität Bielefeld.
`http://www.techfak.uni-bielefeld.de/ags/zellkult`

[2] AG Technische Informatik, Technische Fakultät, Universität Bielefeld.

[3] Currently *Robotics and Embedded Systems*, Institute for Computer Science, Technical University of Munich.
`http://www.atknoll1.informatik.tu-muenchen.de/Zope/tum6`

One explicit intention of the project is to use standard components wherever possible in order to reduce costs. This means that on the robotics side a robot system consisting of a standard robot arm mounted on a mobile platform that is capable of navigating in the laboratory has been built to operate the devices needed for the sample management. The devices on the biotechnological side include a sampling device for filling a sample from a bioreactor into a tube, a pipette for filling aliquots of the sample into other tubes, a cell counter to determine the cell growth and viability, a centrifuge to separate the supernatant and a fridge for storing the supernatant. The entire system is controlled by a computer that logs all data to allow verification. All the devices are standard components or built from standard components, and their suitability for usage with a mobile robot system has been researched.

As a side effect, the use of standard components automatically leads to a very high flexibility. Biotechnological processes are comparatively slow and a single process is only seldomly changed, but only with the flexibility to quickly adapt the robot system to completely other processes it becomes economically reasonable. As with other machines as well, the flexibility determines the commercial success. A secondary goal of the project therefore is to introduce a new level of flexibility in the automation of these processes.

1.2 State of the Art

Automation (greek: *auto* = "self", *matos* = "willing") is about devices that act on their own will – though humans usually take this as machines that facilitate their work. The idea to facilitate the daily work has a long tradition in human history. Ever since the earliest beginnings the homo sapiens has built himself tools to simplify his work, though this is not called automation. Over time these tools became more and more sophisticated, but still they only helped humans in doing their work. Not earlier than during the industrial revolution the first machines were built to really work for humans instead of only helping them working, and this technology has improved ever since.

Since biotechnology is both a rather young and complex science it is by far not yet as automated as, for example, the car manufacturing industry. The first automation attempts focused on semi-autonomous devices, of which biotechnological processes nowadays use a large number. The obvious example is the bioreactor itself, which is connected to a computer that autonomously monitors and adjusts the *physical* cultivation parameters. Automated bioreactors are well established and in use for several decades. A more recent example is the Cedex cell counter used in this project, which evaluates some of the *biological* cultivation parameters (see section 2.2.4). These devices can be very elaborate, as for example the Biomek FX Laboratory Automation Workstation by Beckman-Coulter in figure 1.1, a cartesian robot that is used for automatic liquid management.

Applications of these simple robots can be very advanced and complex. For example, King et. al. have built a system where such a device is integrated into an intelligent robot system that automatically plans and executes experiments to research gene mutations on brewer's yeast (saccharomyces cerevisiae)[5] . In tests their system has proven to be much faster than a human operator performing the same analysis manually, and to develop an experimental strategy that

[5] See [King et. al. (2004)].

Figure 1.1: The Biomek FX Laboratory Automation Workstation by Beckman-Coulter, `http://beckman-coulter.com`.

saves up to 2/3 of the costs[6]. The focus in their application, however, is on the "intelligence" of the system, and not the robot.

All these devices have in common that they do automate parts of a process or a small but very repetitive process, but have to be loaded and unloaded by humans. It is because of this manpower requirements that these devices are called semi-autonomous here.

The automation of the load/unload procedure has led to systems like the TRAC system by TECAN in figure 1.2 or the ORCA robot arm used in the SAGIAN high-throughput screening system by Beckman-Coulter in figure 1.3.

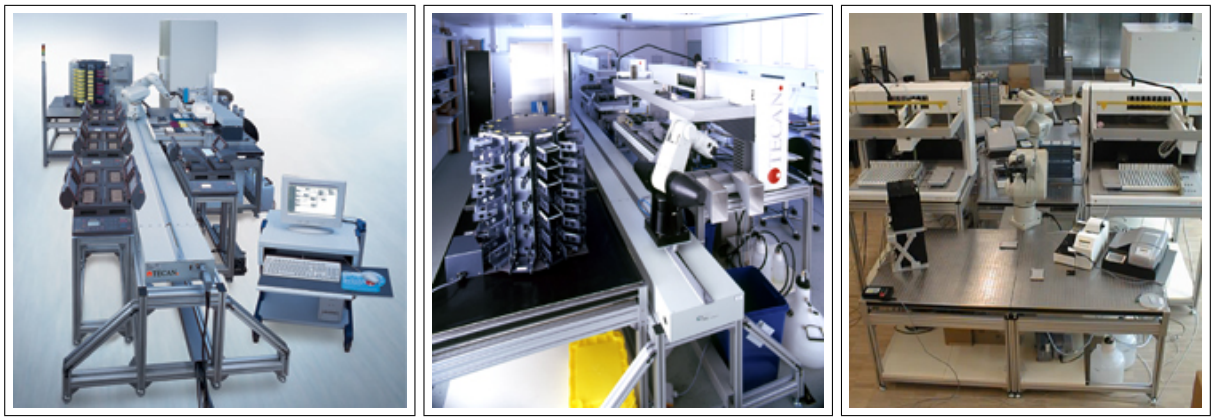


Figure 1.2: The *Tecan Robotic Assay Composer* (TRAC) by TECAN. Images ©2001 Tecan Group AG. All rights reserved. `http://www.tecan.com`

[6]See "Automatische Erkenntnis", c't magazin für computer technik 4/2004, p.46, Heise Verlag, Deutschland.

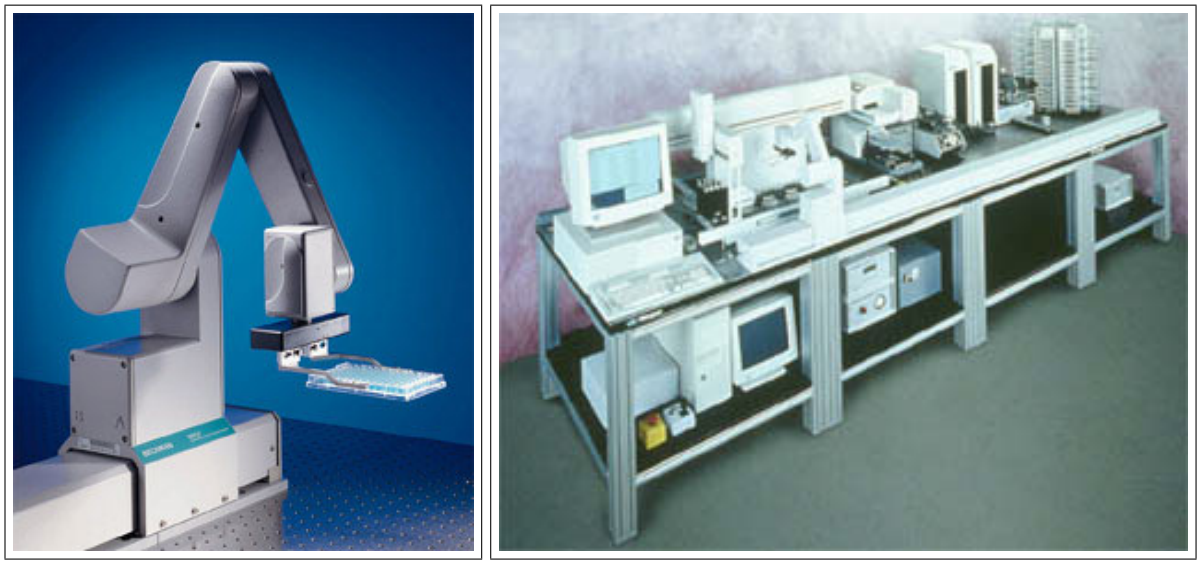


Figure 1.3: The *Optimised Robot for Chemical Analysis* (ORCA, left image) used in the SAGIAN system (right image). Images by Beckman-Coulter, `http://beckman-coulter.com`.

These systems use either stationary robot arms or linear conveyor belts to move the robot arm to the devices, which the arm then loads/unloads, but does not otherwise operate. The components form a robot *workcell* where the devices are grouped so close together that they cannot be used any more by humans for other tasks. Their flexibility is therefore limited to what devices can be placed within reach of the robot. The hardware is highly specialised for the intended tasks, making it very expensive.

Attempts to use standard robot hardware to reduce the costs date back to 1988 when The Automation Partnership[7] (TAP) introduced its "Cellmate" system. The Cellmate system in figure 1.4 is used to automate the cultivation of cells in roller bottles and T-flasks by using a cleanroom version of the Stäubli RX60 arm in a class 100 laminar airflow cabinet. Roller bottles or T-flasks are fed into the system on a conveyor belt, where they are processed following user-defined programs. The "SelecT" system in figure 1.5 was designed in 2001 and is the successor of the Cellmate, meant to increase the number of possible operations. It can handle up to 182 T-175 flasks, either as a single cell line or different ones.

The Cellmate and the SelecT use a cleanroom version of the Stäubli RX60 robot arm (see the introduction of chapter 4). This arm is used to handle roller bottles and T-flasks. Due to their design all devices needed for the operation of the robot have to be arranged in the workcell and cannot be used from the outside for other tasks. Whereas this approach is appropriate for the rapid batches of small scale processes these machines have been designed for, it is inappropriate for large scale production processes. In large scale production processes fewer operations are needed, leading to a comparatively higher idle time of the system. An exclusive use of the devices by the robot would render such a system too expensive, therefore cheaper solutions are needed.

On the robotics side, truly autonomous service robots are also a relatively new field of research. Plain mobile platforms for transportation tasks – known in industry as *automatic guided*

[7]The Automation Partnership, York Way, Royston, Hertfordshire SG8 5WY, UK. `http://www.automationpartnership.com`

Figure 1.4: The “Cellmate” automation workcell using a standard robot arm. Images © by The Automation Partnership. `http://www.automationpartnership.com`

Figure 1.5: The “SelecT” automation workcell by The Automation Partnership using a standard robot arm. Images © by The Automation Partnership. `http://www.automationpartnership.com`

vehicles (AGV) – are in use for a long time, but these early platforms cannot navigate freely because they are guided by visual, magnetic or metal lane markers – they are driving on virtual rails. For simple transportation tasks this is sufficient, but not if true autonomy is desired. Autonomous mobile platforms that do not rely on lane markers are being researched for about 20 years, but are only beginning to leave the university laboratories to be used commercially. One early example of a commercial application is the MORTIMER platform developed around 1998 at the University of Karlsruhe, shown in figure 1.6 together with another platform. MORTIMER has been used in a hotel as a page, carrying baggage and delivering drinks and meals to guests in their rooms[8].

[8] `http://www3.uni-karlsruhe.de/Uni/puk/Veroeffentlichungen/Unikath/Unikath98/morti.html`

Figure 1.6: VIPER & MORTIMER, two service robots developed at the University of Karlsruhe.

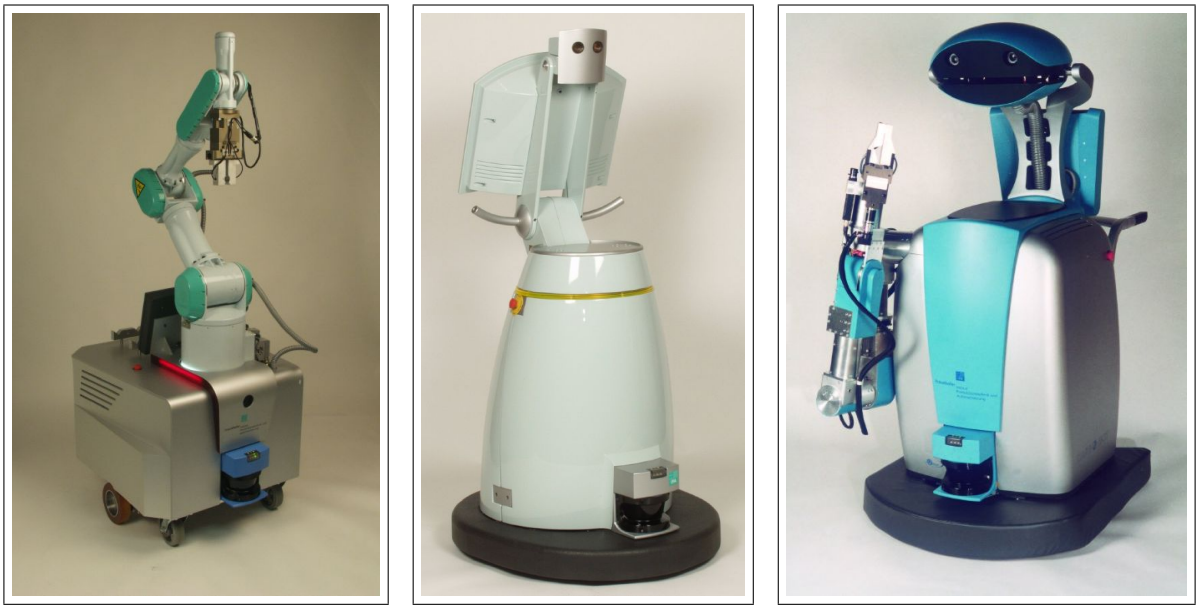
Only in recent years the interest in service robots equipped with arms has risen. The rob@work system in figure 1.7(a) developed in 1998 by the Fraunhofer IPA Institute and used in the ASSISTOR project[9] is one example of such a system. As the project name suggests, rob@work is intended to assist a human worker, for example by – cooperatively – holding heavy objects or by repeating motions it has been taught. It is meant to be an intelligent third hand, but not to fully replace a human worker.

The Care-O-Bot system[10] developed in 1998 by the Fraunhofer IPA Institute shown in figures 1.7(b) and 1.7(c) are not meant to assist a worker with his work, but to assist elderly people with their daily problems. Care of the elderly is – considering the age structure of industrialised societies – increasingly becoming a time- and cost-intensive problem that is especially in Japan addressed by high-tech solutions using robots. These service robots can be used to entertain their users, as a walking aid, to fetch and carry items, to ensure that their users take in their daily medication and even to monitor medical parameters and to call for emergency service if they notice an unusual or urgent condition.

These tasks go beyond plain transportation of objects, and so the Care-O-Bot II has in 2002 been equipped with a robot arm with a gripper. His haptic capabilities are still not comparable to those of a human, but for elderly people the handling of books, bottles and plates can already be of great help. The aim of the Care-O-Bot systems is to allow them to do complex house-keeping tasks like operating a fridge or microwave oven, but the realisation of these goals is still in an experimental phase.

As can be seen at these examples, the trend with service robots focuses on their most commonly known application: To render everyday services to humans. For these types of tasks the systems must be reliable and guaranteed to not do any harm to their environment, but this does not necessarily require an accurate positioning. Safe motion planning and execution can be achieved even with only coarse knowledge about the position, and low-level collision avoidance can be implemented even without any position information at all. With no accurate positioning required for what is esteemed the primary objective, the positioning accuracy of these systems has until now not been a high priority and is therefore not sufficient for objectives like laboratory automation.

[9] `http://www.assistor.de`
[10] `http://www.care-o-bot.de`



(a) rob@work (b) Care-O-Bot (c) Care-O-Bot II

Figure 1.7: Three generations of service robots developed at the Fraunhofer IPA Institute.

Current research aims primarily at making the robots more user friendly, to establish human-like man-machine interfaces controlled by natural speech and gestures and to improve the robot intelligence. In short: The trend is to make the robot act, think, behave and – last but not least – look more like a human, because the physical appearance affects the social acceptance. The ASIMO (*Advanced Step in Innovative Mobility*) humanoid robot by HONDA in figure 1.8 goes clearly in that direction. These systems are meant to be used as entertainment robots, not as laboratory robots – for a laboratory robot the physical appearance does not matter.

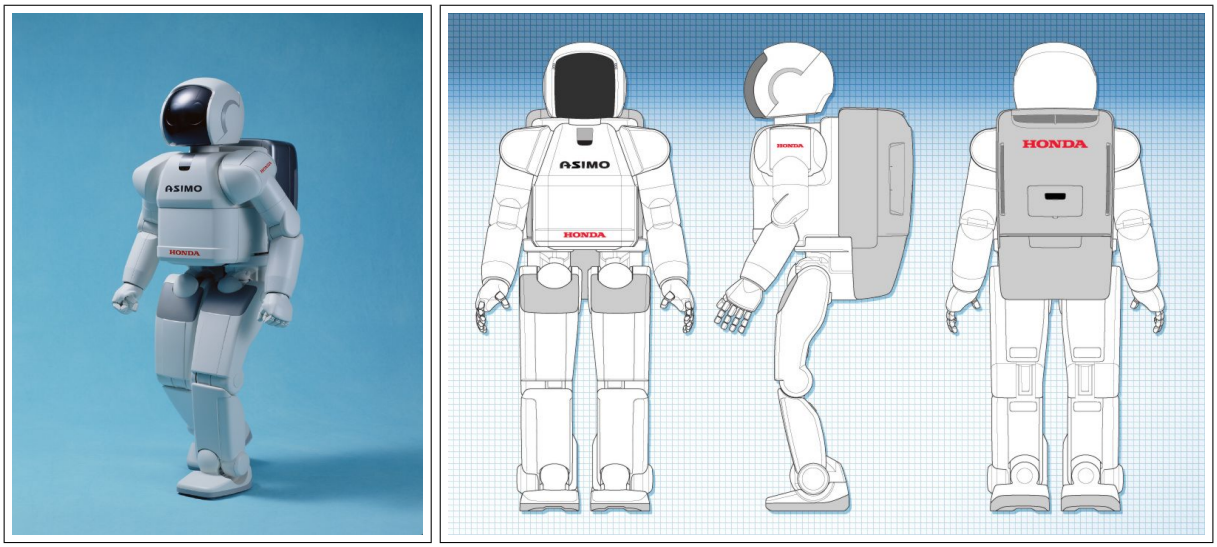


Figure 1.8: The ASIMO humanoid robot by HONDA. `http://asimo.honda.com`

1.3 About this Work

The project in which the biotechnological automation system was implemented employed two researchers over a total of five years: Ms. Dipl-Ing. (FH) Iris Poggendorf has been responsible for the biotechnological part – the process, the verification, the auxiliary devices and the global laboratory automation. Her work is presented in [Poggendorf 2004]. Mr. Dipl.-Inform. Torsten Scherer has been responsible for the robotics part – the design and tests of the mobile robot system and its integration into the laboratory automation system. His work is presented in this thesis.

The aspects covered in this work include a wide range of topics such as:

- Research on and selection (often dominated by availability) of components to build the mobile robot system from,
- design and implementation of control algorithms for the individual components (realised in separate applications),
- design and implementation of a software architecture suitable for integration of the mobile robot system into the laboratory environment (including the accessibility by the laboratory control program and the maintainability of the required parameters) and
- several tests to evaluate the system's performance (single components or the entire system during real cell cultivations).

The primary novelty of this work is the successful combination of a mobile platform and a robot arm to automate laboratory tasks. Mobile platforms and robot arms alone are established automation solutions, but combinations of them are still very rare. They only exist as research objects and not yet as commercial products. As such, they are mostly used for artificial tasks in artificial environments, but not for real world tasks in real world environments. To the best of the author's knowledge, so far no working mobile robot system for automating laboratory tasks exists, and in particular not the biotechnological sample management.

The two major novelties of the established system are its accuracy and its reliability. The accuracy of the positioning – especially of the mobile platform – determines the foundation of the success of the system. A mobile robot that is expected to manipulate standard laboratory devices requires a much higher accuracy than one that is used for entertainment purposes only. The average positioning accuracy of less than 1 cm of the solution presented in this work surpasses that of other systems significantly.

The reliability is – last but not least – the one criterion that determines the acceptance and the commercial success. For example, in the car manufacturing industry a *driver assistance system* (DAS) for detection of pedestrians is considered reliable if it reaches detection rates of 99%[11], simply because the system only assists the driver, who – still remaining in full responsibility – will detect the remaining 1%. For a mobile robot system in a laboratory a success rate of 99% means that it is nearly unusable, because with no supervision there must not be any error at all. However, a real world environment like a biotechnological laboratory means to not have standardised conditions, an explicitly to not have a standardised illumination – a condition that is often required with computer vision.

[11] c't magazin für computer technik 17/2004, p.76, Heise Verlag, Deutschland.

The vision system presented in this work therefore has to tolerate a wide range of illumination intensity. It addresses this problem by using that component of vision that is least sensitive to the illumination intensity – colour information. The newly proposed colour segmentation allows for a very reliable detection of image features, and the model based object recognition further enhances the reliability. The system can still fail under extreme circumstances, but contrary to many other vision systems – especially learning based systems – the limits can be explicitly given and the failure can not go unnoticed. Therefore, the system has achieved a so far unprecedenced success rate.

1.4 Thesis Structure

The robot system consists of several components, most notably a mobile platform, a robot arm and a vision system. In order to keep the complexity maintainable each component has been designed as independently as possible. The thesis follows this idea and also discusses the components separately before presenting the overall results, making up for three major sections. The detailed structure is as follows:

Chapter 2 – "Automating a Biotechnological Laboratory" – describes the laboratory environment and context in which this work is placed. First, it briefly introduces biotechnology and the procedures used for the sample management during biotechnological cell cultivations. Since these procedures cannot be executed by a robot arm alone it then gives a description of the implemented automation system with all auxiliary devices. Based upon these procedures and devices it concludes with the task definition for the mobile robot system.

Chapter 3 – "Accurate Positioning of a Mobile Platform" – discusses the aspects of *localisation*, *path planning* and *motion execution* of the mobile platform. Contrary to other approaches the implemented solutions do not focus on autonomy in exploring unknown terrain or in "intelligent" obstacle avoidance, because the task description does not allow unknown terrain and the system must never be unpredictable. Instead, they focus on precision and predictability, both of which are needed to ensure safety of the system and acceptability by other human personnel in the laboratory. Numerous tests are given to demonstrate the so far unreached precision that is achieved by the implemented approach.

Chapter 4 – "Advanced Robot Arm Control" – discusses the robot arm and tool hard- and software (except the camera and vision software). The low-level aspects of controlling a PA-10 arm at controller level are discussed as well as its integration into RCCL, the Robot Control C-Library, used to access the arm. Results of the newly implemented robot control software are given in comparison with the original vendor software as well as other robots.

Chapter 5 – "Robust Colour Vision" – discusses the hard- and software used in the vision system to fine-position the robot arm. This includes classifying objects and determining their position in an image and is necessary because no fixed object positions can be guaranteed. A very robust and accurate model based approach using colour information is presented together with results from several test.

Chapter 6 – "Realisation and Integration" – discusses some of the components used to integrate the robot system into the laboratory control system described by I. Poggendorf in

[Poggendorf 2004]. The script interface for easy specification of robot actions is presented as well as the state automaton used to ensure low-level safety constraints on these actions. The networked database storing all variable system parameters and also implementing a lookup service for network clients is another of these components. The wrapper classes provided to hide all network details from the user of a network client are finally presented.

Chapter 7 – "Automated Sample Management Results" – demonstrates the robot system's ability to automate the given task by giving photos and results from real test cultivations. These results include biological parameters to demonstrate that the automated system does not affect the culture.

Chapter 8 – "Conclusions & Future Work" – draws conclusions from the experiments and experience with the current robot system and gives a brief survey over some possible extensions and future work.

In addition to these main chapters a collection of documentary chapters are provided in the appendices. These chapters are of minor importance for the thesis because they do not contribute to the primary functionality of the mobile robot system, but may interest a technical reader. The detailed structure is as follows:

Chapter A – "Mobile Platform Documentation" – documents some aspects of the mobile platform more verbosely, like the CAN and RS422 interface to the sensors and the `mobiled` application that implements the control software.

Chapter B – "Robot Arm Documentation" – documents the hardware of the robot arm more verbosely and gives a short introduction into the most prominent features of the *Robot Control C-Library* (RCCL). It shall only motivate the decision to use RCCL in this work, but is not meant to replace the user's guide.

Chapter C – "Vision System Documentation" – documents the hardware of the vision system more verbosely and gives a short introduction into some aspects of computer vision as used in this work. These aspects include several colour models, linear and non-linear image filtering techniques, basic segmentation approaches and some foundations of classification.

Chapter D – "C++ Class Documentation" – documents the major C++ classes used in the various parts.

1.5 Terminology

One issue with any scientific work is the terminology. Due to the ever increasing number of writers an increasing number of interpretations of technical terms exists. These interpretations usually differ only slightly and can in most cases be adjusted by common sense. However, if a reader has strong preconceptions, they can quickly lead to misunderstandings and confusion. Since this work uses several technical terms in a very specific context the most common of these term are defined here in order to avoid these problem.

- The term „(biotechnological) process"– though in a biotechnological context usually used to describe the *entire* cultivation – only addresses the physical appearance of devices and

the sequence of their manipulation done by either a human or a robot. It does not cover any biological or chemical aspects.

- The term „system“ is most often used for the complete laboratory automation system including all devices. It is, however, in a few places also used in local contexts to describe single components covered in these contexts.
- The term „(mobile) robot system“ stands for the compound of the mobile platform, the robot arm, sensor systems and the controlling computer installed on the mobile platform.
- The term „(mobile) platform“ stands for the mobile platform with its sensors, but without the robot arm. Its meaning sometimes includes the controlling computer as far as it is needed to access the platform.
- The term „(robot) arm“ stands for the robot arm with its sensors, but without the mobile platform. Its meaning sometimes includes the controlling computer as far as it is needed to access the arm.
- The term „pose“ stands for the combination of a cartesian *position* and *orientation*.
- The term „position“ usually stands for the translational component of a pose – the *position* –, but is sometimes also used in a more general way.

1.6 A Note on HTTP Links

This work uses HTTP *links* as references in several places. The author is well aware that these links are not fully recognised as references in the scientific world because they tend to be comparatively short-lived and change too often. He does not consider them to be a replacement for official references. He considers them to be additional, but not essential references. Links are used only at places where they provide useful information, but where the absence of that information due to outdated links does not hurt. In the really important cases there is always a postal address or a literature citation in addition to a link.

Chapter 2

Automating a Biotechnological Laboratory

"Life, don't talk to me about Life."

Marvin

Before addressing the real contents of this work – the mobile robot system – some context information has to be given. The environment in which this work took place is a biotechnological laboratory and the task was to automate a certain biotechnological process. For pilot scale setups the size of the laboratory is somewhat limited, but the layout rather non-constant. The specific type of cultivation is also of importance for an automation system. All this imposes several very specific requirements on the one hand side, but allows to make certain simplifying assumptions on the other hand side. Only by taking the physical environment and biotechnological background into account the complexity can be reduced to a manageable point. Without this background some design aspects may appear inappropriate. Therefore, this chapter presents the necessary background information.

Section 2.1 gives a short non-biotechnological introduction into those parts of biotechnology that are needed to understand this work. It defines some basic questions of biotechnology as well as the specific characteristics of cultivating mammalian cells in pilot scale. The reasons for and the procedure of sample management are introduced as well as the specific sample management in the laboratory in Bielefeld. The section closes with a preliminary task definition – not yet specifically meant for a mobile robot system, but more abstract for anyone who has to perform the sample management.

Section 2.2 gives an overview of the architecture of the automation system established in the project. Since the mobile robot system cannot perform the sample management without auxiliary devices this overview basically consists of a description of the involved devices. These descriptions are, however, limited to aspects that are of importance for the mobile robot system.

Section 2.3 gives the official task definition for the mobile robot system, according to the requirements as imposed by the auxiliary devices. This includes abstract specifications of capabilities as well as numeric values for accuracies where they are known.

Section 2.4 gives an overview over the architecture of the mobile robot system devised, designed and built in this work. This robot consists of standard hardware components, but the software had to be newly developed to meet the task definition. The individual hardware components and their software are discussed in subsequent chapters.

2.1 Biotechnology

Biotechnology – from a layman's viewpoint – is the science of cultivating cells, and involves both biological aspects of cells and technological aspects of devices. The cells have to be kept in a watery solution, so cultivating them means to keep them in any kind of storage ranging from small petri dishes over t-bottles (20-50 ml), spinners (100-250 ml) and super-spinners (1000 ml) to bioreactors (20 l and more). The purpose of these cultivations is either to obtain the cells themselves or to obtain some product from their metabolism, for example pharmaceuticals.

2.1.1 Cells Compounds versus Single Cells

If the cells themselves are of interest, for example when growing artificial skin tissue for transplantation to people with severe skin burns, this means growing larger compounds of cells of a single specific type. It is not yet possible to artificially grow complex compounds of cells of different types, like entire organs, though the future might bring that soon.

In any case growing compounds of cells means to let them organise themselves in their preferred structure without external influence. This narrows the number of possible storage containers to grow them in down to containers where they can be fixedly kept in place, for example petri dishes. Each such container requires a different way to provide nourishment to the cells, for example spinners usually use bubble-gassing whereas super-spinners use membrane-gassing. Petri dishes, on the other hand side, are surface-gassed. All this makes up for very specific setups for individual cell types. Since cell compounds are not the scope of the biotechnological group in Bielefeld this work does not cover these issues any further, but instead focuses on single cell cultivations.

Cultivating single cells means one can let them float freely in a nourishing broth, which simplifies the process and allows for a much larger production scale than petri dishes. It is this type of cultivation that uses bioreactors, which exist with capacities of up to several 1000 litres.

2.1.2 Cultivation of Mammalian Cells in Pilot Scale

The "cultivation of mammalian cells in pilot scale" is about growing single cells. The question of growing single cells leaves out most aspects of chemical or physical interaction with other cells, but the aspects of the internal cell metabolism remain. In order to keep that metabolism under optimal conditions a number of external parameters of the environment in which the cells are supposed to live have to be controlled. The most important of these parameters include

- the p_H value indicating the acidity of the broth (in terms of a watery solution),
- the ambient temperature ϑ of the broth,
- the p_{O_2} value indicating the amount of oxygen dissolved in the broth and
- the amount of nourishing substances in the broth.

Apart from the amount of nourishing substances in the broth these parameters are called *on-line* parameters because they can be measured continuously and *in-situ*, that means inside the reactor and while it is running. Other – more indirect – on-line parameters include

- $V_{air}[l/h]$, the amount of air (oxygen) per hour needed to maintain or raise the p_{O_2} value,
- $V_{N_2}[l/h]$, the amount of nitrogen needed to reduce the p_{O_2} value and
- $V_{CO_2}[l/h]$, the amount of carbon dioxide needed to increase the p_H value.

All these parameters are controlled in real-time by the controller of the bioreactor. They can be used to detect abnormal situations by comparing their course with the expected course gained by prior experience. An infection can for example be discovered by an abnormal rise of the O_2 consumption.

However, on-line parameters only offer an indirect view of the cells' health. For example, controlling the temperature of the broth is usually done by heating the bioreactor's jacket with steam or electricity. This, however, means that the temperature distribution in the broth is not uniform. Near the jacket the broth will be hotter than in the middle of the reactor, and cells that stay close to the jacket for too long may get damaged. Stirring the broth to prevent this from happening is only a compromise because it introduces the problem of physically damaging the cells with the propeller of the stirrer. On the other hand side stirring is absolutely necessary to ensure a uniform gas distribution if bubble gassing is used.

As a consequence, a lot of different heating, stirring and gassing systems have been manually optimised for each individual cell line, but still one cannot guarantee the culture's health from their parameters. Only by analysing the culture directly information about its health can be gained.

2.1.3 Sample Management

The purpose of the sample management is to directly measure the cell density and viability of a culture. Approaches to design appropriate sensors to be plugged into the bioreactor directly do exist, but none of them have passed the experimental state yet. State of the art therefore still is to use external devices. This means that the reactor has to be opened and a sample has to be taken and analysed, which is why the parameters obtained from this step are called *off-line* parameters.

The sample management as performed at the University of Bielefeld is divided into the following steps:

Taking a Sample

Opening the otherwise strictly sealed bioreactor to take a sample presents a sterility risk and therefore extra caution has to be taken that the bioreactor is not contaminated. Depending on the type of sampling valve/device used this means that a more or less large amount of equipment has to be sterilised before/after the sampling process.

In our case steam sterilisable valves directly attached to the reactor and covered with a sterilisation sleeve are used. To take a sample the sterilisation sleeve is removed, a tube is held to the valve, the valve is opened and the sample is filled into the tube. After removing the tube from the valve the sterilisation sleeve is put back and the valve sterilised by pressurised steam for several minutes.

Note that this procedure means that the valve and therefore the reactor is kept sterile, but not necessarily the tube and therefore the sample itself. Since the subsequent analytical steps are rather fast compared to the growth rate of bacteria, the remaining risk of contaminating the sample does not really present a danger for a cultivation under "pilot scale" rules. For a "real" production stricter rules apply. Over all, sterility is a feasible issue in sample management.

Analysing Cells

Another issue is the one of analysing the sample to count the cells and judge their viability. The most commonly known method for analysing cells is the *trypan blue dye method*. Using this method the sample is dyed with trypan blue and analysed under a microscope. Two major cases are distinguishable:

- If a cell is alive the dye only attaches to the hull. In this case the cell will be visible as a dark circle.
- If a cell is dead it probably has a broken hull and the dye can enter the cell. In this case the cell will be visible as a solid dark ball, therefore being easily distinguishable from a live cell.

Using this method a number of subsamples of the sample are analysed and an average cell count (cell density) and cell viability rate can be given. If performed by a human operator, this is a time-consuming and error-prone task because humans, depending on their training and fatigue, will miscount or misclassify the cells and thus introduce noticeable errors. What is even worse is that different people may use different thresholds for classifying ambiguous cells, thus making the process of analysing the cells indeterministic. As a result the state of a culture may not be correctly observed and a potentially dangerous situation may be noticed too late.

A more deterministic approach would be of great benefit here. The idea is that even if such an approach introduces new errors, it is far more advantageous to have a knowingly constant – even if erroneous – system than to have an indeterministic system where the error changes from measurement to measurement.

Storing a Cell Free Aliquot

The final step is to store a cell free aliquot of the sample in a freezer for later analysis of amino acids. This involves separating the cells by either filtering or – as in our case – centrifugating, filling the cell free supernatant into a different tube and storing this tube in a freezer. The broth will not deteriorate if frozen and can therefore be analysed at any time later on.

In addition to checking the culture's cell density and viability the sample management is also used to determine the optimal harvest time. The exact time of harvest is not so much determined by the time since the start of the cultivation, but more by the growth rate of the culture and the absolute cell density. In one-shot cultivations the cells consume the nourishing substances in the broth with increasing speed, and once they are consumed harvest should take place because the culture will die shortly afterwards[1]. Due to the basically slow growth rate of mammalian

[1]Harvesting for one-shot cultures means to empty the reactor, separate whatever produce is desired from the broth and dispose of the rest. Since this is specific to the culture and has got nothing to do with the mobile robot system it is not further described here.

cells the exact time may vary by several days, depending on the actual growth rate. Only by using sample management the actual growth rate can be established, and therefore the optimal harvest time.

Summarised, the goals of the sample management are:

- to examine the culture's *cell density* and *viability*,
- to determine the optimal *harvest time* and
- to allow later analysis of the cell broth.

After this definition of the sample management process two specifics of the *"Cultivation of Mammalian Cells in Pilot Scale"* must be given because both of them strongly affect any automation system. These specifics are the reproduction rate of mammalian cells and the physical size of the laboratory.

Reproduction Rate

The reproduction rate of mammalian cells is rather low. Bacteria like the often used *escherichia coli* duplicate about every 20 minutes. The mammalian cells used in biotechnology, on the other hand side, need about 20 to 24 hours to duplicate. This means that a contamination with bacteria can outgrow a cultivation in a matter of hours, which might lead to the idea of checking the culture rather frequently. Unfortunately taking a sample too often means a significant and disturbing loss of broth for small cultures, as well as an increased risk of contamination in general. The conclusion is that contamination with bacteria is best checked with high-speed on-line parameters and not with sample management. Taking one sample every 4 to 6 hours is usually considered enough. The conclusion is that for a mobile robot system speed is not the major issue.

Pilot Scale Processes

The term "pilot scale" refers to processes that are used to evaluate new cell lines, new process types or simply to produce cells to inoculate a larger process. Developing new processes or scaling them up is not at all trivial. Using a bioreactor of ten times the size with ten times the gas supply and the capacity of heating and stirring ten times the amount of broth does in general not lead to ten times the number of cells. A lot of parameters are non-linearly correlated to the process size and therefore a lot of process changes may have to be done before finding the optimal parameter set.

For a robot system this means that the process cannot be assumed to be static. The equipment is more likely to change or be moved around the laboratory than for fixed production processes, and a robot system must therefore be flexible enough to follow these changes. On the other hand it also means that the size of the space required for the process is comparatively small. The equipment can be assumed to be gathered in only one room, so that the mobile robot system's path planning capabilities can be kept simple.

A normal one-shot cultivation of mammalian cells like in the 20 l or 100 l bioreactors in the laboratory in Bielefeld takes up to two weeks, whereas continuous cultures at productions plants may run for several months. Since the on-line parameters are automatically controlled by the bioreactor the sample management is usually the only task where human personnel has to be present. This includes both nights and weekends and therefore presents a high cost factor.

In addition to this, using human personnel to visually count and classify the cells introduces a statistical error in the measurements that should best be avoided. It is because of these two drawbacks of current manual sample management that the idea to design and establish an automated system has been developed.

2.1.4 Preliminary Task Definition

An automation system that is supposed to overcome the drawbacks of manual sample management is required to operate at any time of day or night without fatigue, but instead with constant and verifiable quality. It has to perform all manipulations as required by the sample management as described in section 2.1.3, which – in very abstract words – means it has to

- take a sample from the bioreactor (while maintaining sterility),
- determine the cell density and viability rate of the sample (in the most deterministic way),
- produce a cell free aliquot of the broth and
- archive that in a deep freezer.

Even a human operator cannot do these tasks alone, but instead requires auxiliary devices. Because of the project goal of using standard equipment the automation system is supposed to use the same devices. From an economic point of view this has the advantage that existing devices can be used and thus money saved, but from a robotic point of view it has the disadvantage that a mobile robot in the automation system has to manipulate a number of devices that were originally designed to be manipulated by humans. Without anticipating too much details of the devices as presented in section 2.2 the robot is required to be able to

- move freely in the laboratory to dock to devices,
- pick and place 50 ml NUNC tubes and 1 ml CEDEX tubes,
- fill a sample from the bioreactor to a 50 ml NUNC tube by using a sampling device,
- decant aliquots of the sample or the supernatant to other tubes by using a pipetting device,
- count the cell density and viability by using a cell counter,
- separate the cells from the cell-free supernatant by using a centrifuge and
- archive the (barcode labelled) supernatant in a deep freezer.

2.2 The Automation System Architecture

Many of the sample management tasks cannot be directly performed by either a human operator or a mobile robot system alone, so a couple of auxiliary devices have to be used. It is only the combination of the mobile robot system with these devices that makes up the automation system. With the intention to use standard components this means that the automation system basically consists of

- the laboratory itself,
- the mobile robot system,
- a bioreactor that contains the broth for the cultivation,
- the sampling device(s),

- the pipetting device,
- the cell counter,
- the centrifuge and
- the fridge
- a barcode scanner for reading the barcode of the 50 ml NUNC tubes that are to be archived in the fridge and
- several storage racks for 50 ml NUNC tubes with and without barcode and for 1 ml Cedex tubes.

These components have been arranged in an architecture as in figure 2.1, which has two central points: the *laboratory control program* (LCP) and the mobile robot system. The LCP implements the top-level control and coordinates the sample management cycle. It sends commands to the devices to make them perform actions and reads back status information and measuring data. The mobile robot system is "just" one of these devices, although the most important one. It autonomously performs all those tasks which the LCP cannot make the devices do on their own. It is described in more detail in subsection 2.4.

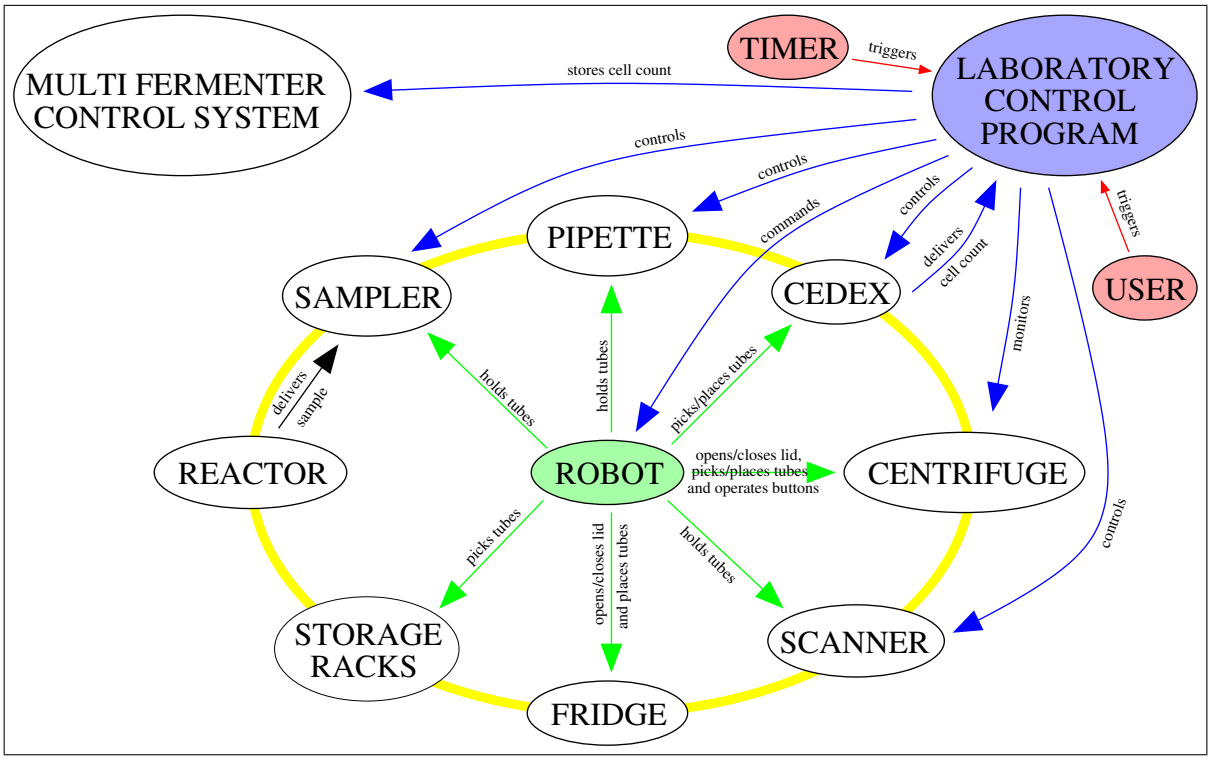


Figure 2.1: The laboratory automation system architecture.

In the following those system components that contribute to the complexity of the mobile robot system will be described, but only up to the level needed to understand how the robot interacts with them. The question of how they are otherwise accessed or controlled or the question of their logical integration in the automation system is not part of this work. These questions, together with the LCP, are discussed by I. Poggendorf in [Poggendorf 2004].

2.2.1 The Laboratory

The least obvious component of the automation system is the laboratory itself, an overview of which is given in figure 2.2. The reason for it being least obvious is that it is usually not considered a “device” of the automation system, but since it does affect the system it must be mentioned. The laboratory is in theory the most easily exchangeable component, however, a few constraints apply.

Figure 2.2: The cell culture laboratory at the University of Bielefeld.

The laboratory in Bielefeld consists of only one room the robot has to operate in, so the questions of passing doors, using ramps or escalators have not been dealt with – the situation is explicitly two-dimensional. On the other hand side the laboratory is quite packed with equipment, some of which – most notably the reactor – cannot be freely placed because it must be attached to fixed installations. In any case a large area in the centre of the laboratory remains clear and can be used for navigation. This means that the robot needs path planning capabilities to navigate around the equipment, but that these capabilities can assume plenty of free space and do not need to include critical situations like tight passageways.

There are no bumps like safety barriers preventing liquids from flowing around in the floor inside the room, but three drains and one lid covering a maintenance chamber. In addition, rubber tubes may be lying on the floor near some devices. It is not possible to move the robot around these objects, because treating them as obstacles would mean that the access to large parts of the laboratory would be blocked. When driven over they can cause the wheels to slip, so the robot control software has to be aware of this.

Contrary to an office environment, the laboratory does not primarily consist of straight walls or corridors. Not to mention cupboards and table or chair legs, most of the walls are obstructed with sometimes bizzarely shaped objects like radiators and pipes. Again, some of these objects are not fixed, but may be moved around. When detected with a distance measurement sensor they can not easily be matched with a map, and therefore do not contribute to the positioning of the mobile robot. Other means of localisation must be used, for example artificial visual landmarks. The laboratory must provide enough possibilities to attach such landmarks in places where the robot can see them from the largest possible number of positions. This means that the laboratory must not be too packed with non-fixed equipment onto which no landmarks should be attached.

2.2.2 The Sampling Device(s)

As with manual sampling, the sample is filled into a 50 ml NUNC tube sealed with a membrane to prevent aerosols from escaping during the filling process. To fill the tube several sampling devices have been built, but from standard components (x/y cartesian robots, see figure 2.3). These sampling devices use different actuation and interface techniques, but are equivalent concerning their biotechnological and robotical aspects.

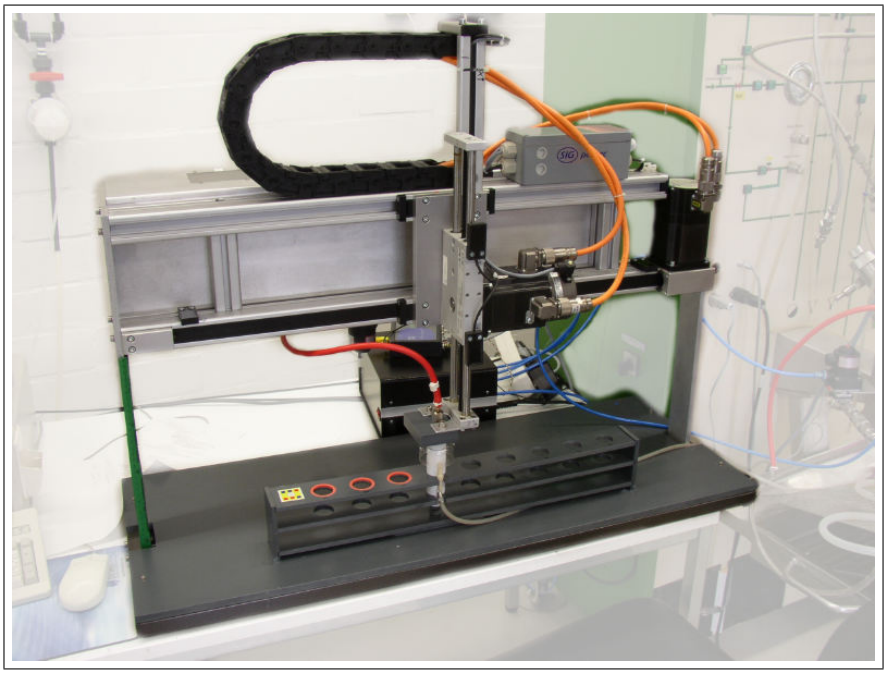


Figure 2.3: One of the sampling devices.

Each sampling device is attached to a bioreactor by means of several valves and pipes that can be sterilised with pressurised steam. Before filling a sample this pipe system is flushed with broth to flush out condensate that may have settled during the cool-down after sterilisation. If the pipes were not flushed with broth the sample would otherwise be diluted by the condensate, leading to a decreased cell count. The samples are then filled by piercing a needle through the membrane and opening the main valve for a specified amount of time.

To fill a sample into a tube the robot has to place an empty tube into the slot of the storage rack, re-grasp the tube and fixate it while the sampler is working. This is because of the piercing of the needle, which would pull out the tube from the rack when the needle is removed afterwards. After the sampler has finished the robot can retake the tube from the sampler again. In more detailed words, the robot has to be able to

- go to the sampler,
- find and fine-position over the sampler,
- find and fine-position over the slot,
- place the tube in the slot,
- fix the tube while the sampler works and
- re-grasp the tube.

2.2.3 The Pipetting Device

At several points during the sample management aliquots of the sample have to be filled from one tube into another. As with manual sampling, the robot uses a pipette for this task, but since it cannot grasp an ordinary pipette a special pipetting device has been built (see figure 2.4). This pipetting device consists of two standard components: A motor driven syringe and a motor driven multiport valve. Similar to the sampling device the pipes of the pipetting device have to be cleaned and sterilised between two operations, but this time alcohol is used for this. A bottle with alcohol must be attached to the pipette as well as a bottle taking the waste – hence the multiport valve.

Figure 2.4: The pipetting device, together with some of the storage racks.

A reservoir holding 1 ml directly above the needle prevents cells from sedimenting in places where they would not be flushed out again. If they would – for example – sediment inside the syringe, this would dilute the sample, leading to a decreased cell count.

To fill aliquots from one tube into another the robot first holds a filled tube under the needle, the pipette then sucks in the required amount of liquid, the robot then removes the filled tube and holds an empty tube under the needle and the syringe finally ejects the liquid into the tube. When using the 50 ml NUNC tubes the robot must control the depth how far the needle pierces into the tube because for freshly filled samples with it has to draw the aliquot from the bottom of the broth, whereas for centrifugated samples it has to draw the aliquot from the cell free supernatant at the top of the broth. In more detailed words, the robot has to be able to

- go to the pipette,
- find and fine-position over the pipette,
- hold a (NUNC or CEDEX) tube under the needle, allowing for an offset how deep the needle should puncture through the septum into the tube (in case of NUNC tubes) and
- remove a tube from under the needle.

2.2.4 The Cell Counter

Counting the cells and determining their viability is the central part of the sample management. The usual way to do this is to dye the sample with trypan blue and analyse it under a microscope. If a cell is alive the dye only attaches to the hull, leading to a visible dark circle outline. If a cell is dead it probably has a broken hull and the dye can enter the cell, leading to a visible solid filled circle.

The Cedex cell counter by Innovatis[2] is a device that automates this procedure. In figure 2.5 the Cedex can be seen with the AutoSampler AS20 attached – a carrousel holding several "Cedex tubes" for batch measurements. Whereas the Cedex is originally designed to be used by a human operator it can in combination with the AS20 be fed tubes by a robot.

Figure 2.5: The Cedex cell counter.

To analyse a sample an aliquot of exactly 1 ml is filled into a Cedex tube using the pipette, which is then inserted into the frontmost slot of the carrousel. The AS20 then rotates the carrousel so that the piston in its back can lift the tube into the docking port of the Cedex. After the analysis the Cedex tube is ejected again and removed from the carrousel. The operation of the AS20 and the Cedex itself are not controlled by the robot, but by the LCP. The robot is only needed to feed the Cedex tube to the carrousel. In more detailed words, the robot has to be able to

- go to the cedex,
- find and fine-position over the CEDEX,
- find and fine-position over the slot in the carousel and
- grasp or place a tube in the slot.

[2] Innovatis AG, Meisenstraße 96, D-33607 Bielefeld, Germany. `http://www.innovatis.de`

2.2.5 The Centrifuge

Since the cells, even when in a non-tempered and non-gassed tube, will continue to live for a while after being taken from the reactor their metabolism will affect the broth. To allow a precise biochemical analysis of the sample a cell free aliquot must therefore be archived. Apart from filtration the most common way to obtain such an aliquot is to centrifugate the sample to sediment the cells.

To centrifugate the sample a 50 ml NUNC tube is inserted into the free slot of the cage inside the centrifuge. Since the cage does not stop in a predetermined position and/or a human operator may have used the centrifuge since the last automated sample management cycle any position of the slot must be allowed. After the tube is inserted the lid is closed. To start the centrifuge the “start” button is pressed. After the centrifugation is completed the “lid” button is pressed to release the lid locking mechanism. The lid is then opened and the tube taken from the slot in the cage. Again, any position of the slot must be allowed.

The centrifuge in figure 2.6 used for this is a standard device and explicitly not designed to be operated by a robot. This means that the runtime and rotation speed are manually programmed in advance, but for everything else the robot must act like a human. Since with its fingers it cannot pick or place the tube for an arbitrary slot position it may have to rotate the cage “manually” until the slot is in a proper position. If this happens when placing a tube into the slot, the tube must temporarily be deposited into a local storage rack because the robot cannot rotate the cage with a tube in its fingers.

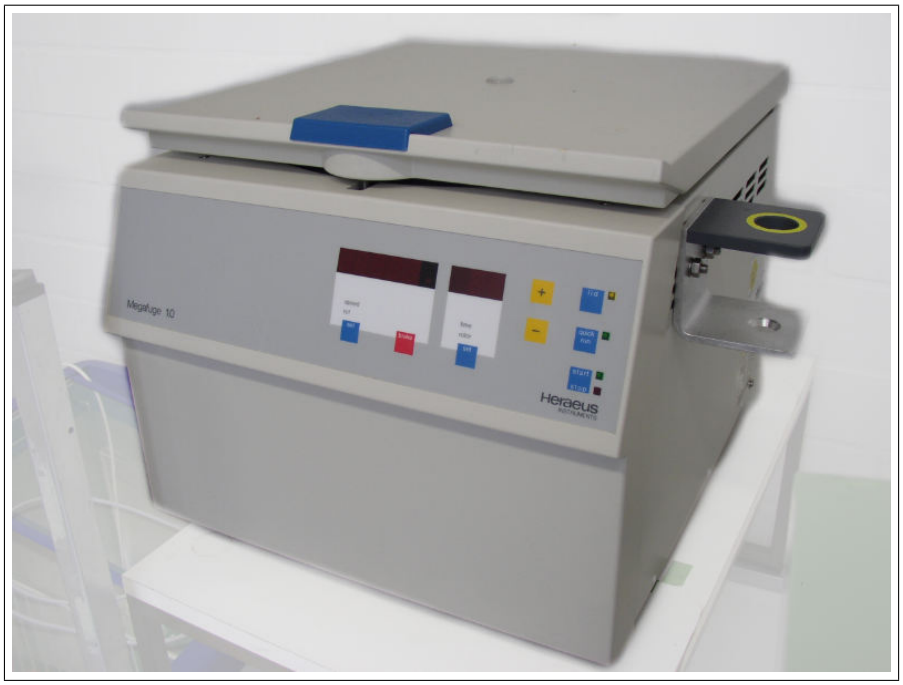


Figure 2.6: The centrifuge.

When closing the lid by pushing it so that it falls down it cannot be guaranteed that the locking mechanism really snaps. The robot therefore uses a force-aborted test motion to verify that the lid is really locked close. If it is not, it uses an additional force-controlled motion to press onto the lid until it detects a spike in the forces that indicates that the locking mechanism has snapped. Similar safety applies to opening the lid. In more detailed words, the robot has to be able to

- go to the centrifuge,
- find and fine-position over the centrifuge,
- open the lid by
  - pressing a button to unlock the lid,
  - checking if the lid is really unlocked and
  - opening the lid with a circular motion,
- locate the position of the empty/full slot in the cage,
- rotate the cage if necessary to reach the slot by
  - placing the grasped tube in a temporary rack (if applicable),
  - reaching down with the rubber finger onto a pad of the cage,
  - rotating it piecewise to the desired position and
  - regrasping the tube (if applicable),
- pick or place a tube in the slot,
- close the lid by
  - closing the lid with a circular motion,
  - checking if the lid is locked and
  - pressing the lid down if it is not and
- start the centrifuge by
  - pressing a button.

2.2.6 The Fridge

The second part to keep the sample from deteriorating besides centrifugating off the cells is to store the sample in a freezer. Only if frozen it can reliably be analysed at a later time. A standard freezer with a sliding top lid is used for this (see figure 2.7), together with barcode labelled 50 ml NUNC tubes. The barcode is used to double-verify that the association between the sample and the corresponding cell count can be established.

To freeze a cell-free sample the robot opens the sliding lid by moving behind the handle and pushing it sideways. This operation is force-controlled to verify that the lid does not get stuck before the end of the motion. Since there is no temporary storage rack it can only be done when the gripper is empty. The robot can do this for example while it is waiting for the centrifuge to stop – due to the top lid the fridge can be open a few minutes without loosing too much cold. After filling the sample into an empty tube and reading its barcode the robot goes back to the fridge and places the tube in one of the free slots of the storage rack inside it. It finally closes the lid again by applying the reverse sideways motion to the handle of the lid. In more detailed words, the robot has to be able to

- go to the fridge,
- fine-position over the fridge,
- open the sliding lid by

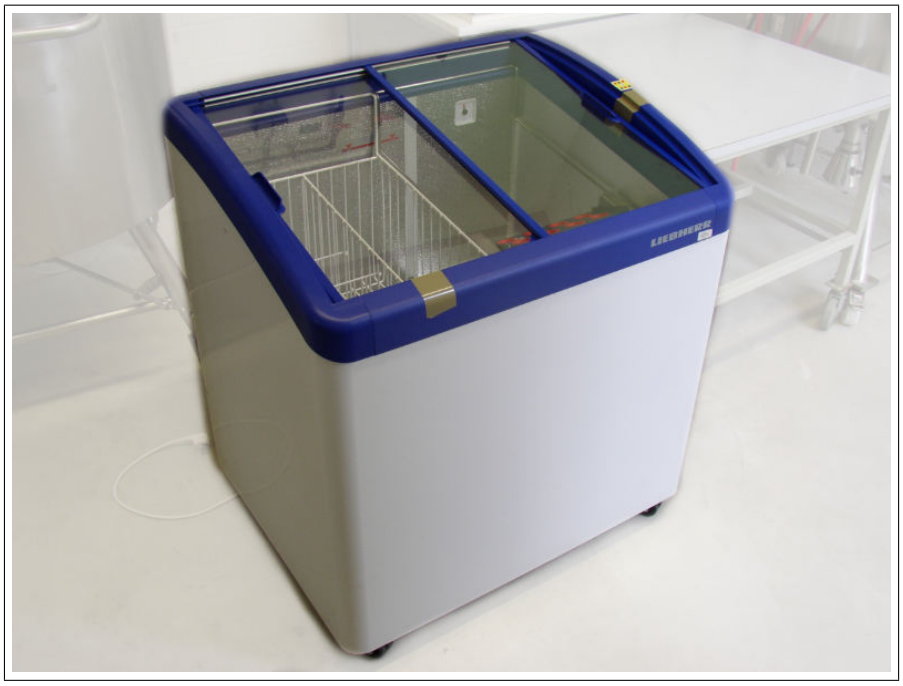


Figure 2.7: The Fridge.

  - pressing on the lid behind its handle,
  - moving sideways while controlling the contact force and
  - watching out for excess forces indicating errors,

- find and fine-position over a free slot,
- place the tube in the slot and
- close the sliding lid again.

2.3 Task Definition

Based on the above descriptions of the devices needed for the sample management the preliminary task definition from subsection 2.1.4 can now be refined. Basically, the mobile robot system has to be able to operate all devices, which are distributed in the laboratory. The detailed requirements of the major components are as follows.

The Mobile Platform

The mobile platform has to bring the robot arm to the devices close enough to allow it to manipulate them. In order to do this is must...

- Compute its own position in the laboratory with a precision of less than 1 cm. Since the measurements involve sensors with a basic accuracy lower than that the fusion of several sensors is mandatory.
- Plan piecewise straight line paths around known obstacles in a map. Since the motion time is not a critical factor and the control software should be kept simple piecewise straight line paths are considered sufficient. This means that only straight line translations and rotations on the spot are required, but not complex curves like splines.

- Execute the motions to arrive at the target with an error of no more than 1 cm. It is this requirement in particular that presents a challenge that has so far not been met by other platforms.
- Avoid collision with unexpected obstacles while moving. Since the laboratory can be considered to be a rather tidy environment unexpected obstacles are assumed to be humans standing around the robot. They are likely to move away on their own, so it is sufficient to stop and wait until they do so.

The Robot Arm

The robot arm has to pick and place tubes, carry them between devices and manipulate some devices. In order to do this is must...

- Determine object coordinates with an accuracy of less than 1 mm. With only very primitive tactile sensors this automatically means to use a vision system. The requirements of this vision system are listed below.
- Measure contact forces to control offsets that cannot be compensated by the vision system. A standard wrist mounted force/torque sensor can be used for this.
- Perform joint and cartesian interpolated motions with an accuracy of less than 1 mm. This means that the time lag between desired and observed positions as introduced by the entire arm control loop including the trajectory generator must be very small.
- Allow real-time modifications of motions in the trajectory generator to implement force control. Without real-time modifications only look-and-move strategies are possible, which are not applicable to force control. Since a few manipulations absolutely require real-time force control the arm control software must support this.
- Grasp 50 ml NUNC and 1 ml Cedex tubes. To allow more stable grasps than a two-point contact this means that the gripper fingers must have circular cut-outs of the appropriate radius, possibly covered with a rubbery material. Due to the great difference in size of the two types of tubes this leads to a non-trivial finger design.
- Perform peg-in-hole type motions to place tubes in slots of storage racks[3]. For the 50 ml NUNC tubes these motions can be force-controlled because the bevelled edges of the slots and the cone shape of the tube's bottom allow to compensate positioning errors on-line.. For the 1 ml Cedex tubes this is not possible because their bottom is not cone shaped. To place them into the slot on the carrousel of the Cedex a very accurate "blind" positioning is therefore needed, and the force sensor can only be used to implement safety aborts.
- Press buttons on the plastic foil keyboard of the centrifuge. Since the sensitive area of a button on a plastic foil keyboard is very difficult to hit and the pressure point very difficult to detect a rubber finger tip is best used for this. It allows both more deviation off the exact button centre and the exact pressing depth. Note that a button on an ordinary keyboard would be much easier to press.
- Perform circular motions, either force-controlled to rotate the cage of the centrifuge or not force controlled to open/close the hinged lid of the centrifuge.

[3] Picking tubes from slots is comparatively trivial and not listed here.

- Perform force-controlled straight line motions to open/close the sliding lid of the fridge.
- Support position equations as target descriptions to allow to conveniently implement the above motions while still not requiring to manually recompute all targets when a single position is changed.

The Vision System

The vision system has to detect the position of objects (devices and tubes) because due to inaccurate initial measurements and human influence on the setup fixed positions can not be assumed[4]. In order to do this is must...

- Provide positional information with an accuracy of less than 1 mm. This means that it cannot rely on features determined by only very few image pixels, but must consider larger regions to allow averaging.
- Classify objects with the utmost robustness. Contrary to common classification tasks like sorting letters according to the zip code on them the detection of the devices and their position must never fail because a failure cannot be corrected at a later stage. If not avoidable in the first place, an error must under all circumstances be detectable to at least safely stop the system.
- Operate robustly without a standardised illumination. Since the system has to work in a standard and potentially large laboratory installing a special illumination is not feasible[5] It must therefore use whatever illumination is present, which means that it must tolerate a larger variation of the illumination as is common.

Apart from these requirements, the robot system must integrate itself seamlessly into an environment of computers and devices, which involves a number of networking issues. Its control software must contain wrappers that hide as many of these issues from the user as possible. Since the system is to be used by non-expert personnel all specifications of positions, motions and other parameters have to be easily modifiable and must not require too much knowledge about robots.

2.4 Robot Architecture

The mobile robot system in figure 2.8(a) that is devised, designed and built in this work is the most obvious component of the automation system. It consists of standard hardware components serving different purposes. The hardware has been explicitly chosen with the task definition in mind, but still all the software had to be newly developed to meet the requirements. The most important of the components are:

- A mobile platform as described in chapter 3. Its purpose is to move the robot arm around the laboratory in order to bring it close enough to other devices so that it can perform its manipulations. It is able to locate its position in the laboratory and plan a collision-free

[4] In case of the centrifuge not stopping at a predetermined position this is true even without human influence.

[5] The only partly reasonable alternative would be to mount lamps to the robot, but this introduces questions of power supply, complicating the tool design and hard shadows and/or brightness gradients due to spot lights. Such an approach has not been considered.

path to any target, including avoiding collisions with unknown obstacles during motion execution. To allow it to do this, it is equipped with several sensors. The positioning accuracy achieved with the control software presented in this work surpasses the previous commercial solution significantly.

- A robot arm and tool as described in chapter 4. Its purpose is to pick, place and carry different types of tubes, hold them to devices and to manipulate those devices that cannot be controlled by other means. It is able to perform several motion types given by position equations, detect collisions with objects and control forces applied to them during compliant motions. To allow it to do that, the tool is equipped with an electric parallel yaw gripper with specialised fingers, a wrist-mounted force/torque sensor and a microhead colour camera used by the vision system. The new control solution presented in this work allows more flexibility than the commercial solution and is significantly more accurate.

- A vision system using the camera on the arm tool as described in chapter 5. Its purpose is to compensate for the displacement of objects caused by inaccurate initial measurements or human influence of the setup during runtime. To allow it to do that, it uses a colour micro-head camera to detect characteristic patterns consisting of coloured regions from either natural features or artificially attached labels. This new approach is more tolerant against changes of the illumination than other approaches and therefore more reliable.

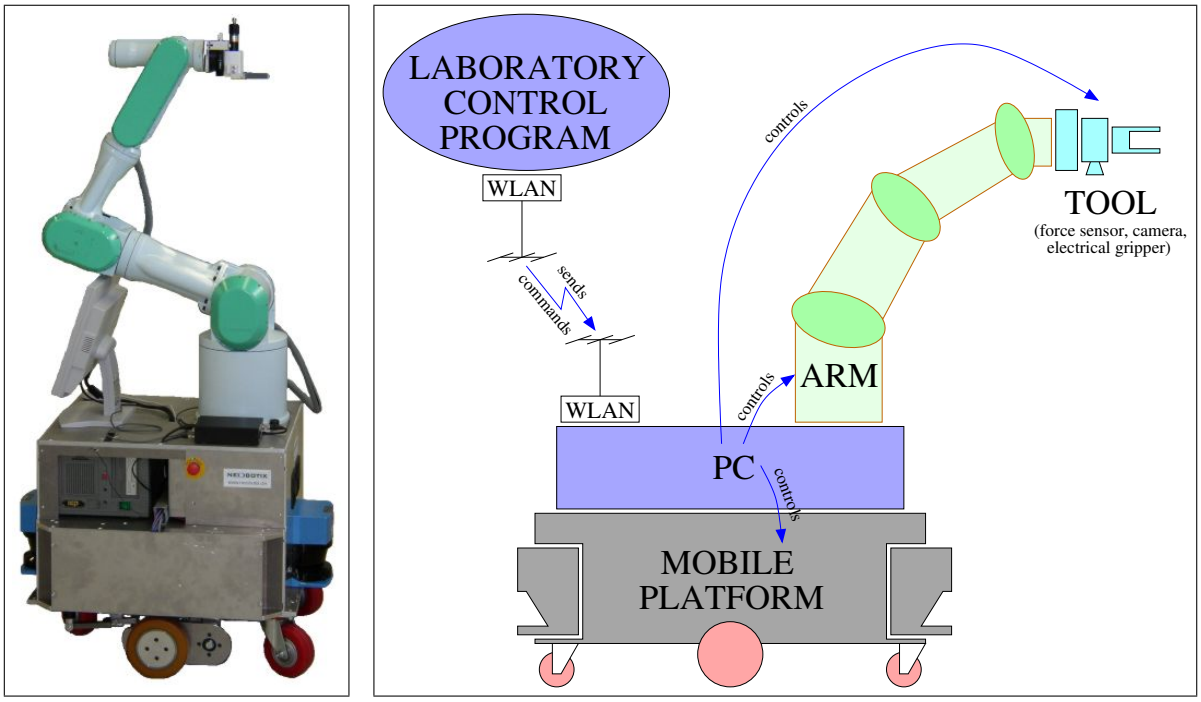


Figure 2.8: The mobile robot system, photo and architecture.

These components are – in order to simplify things – treated as separately as possible. This results in a robot architecture as given in figure 2.8(b). Although all components are controlled by the on-board PC, treating them separately not only affects the hardware, but also the software. Each component is accessed by its own algorithms implemented in its own piece of

software, in particular the mobile platform and the robot arm are controlled by two entirely separate programs. Due to a lacking real-time connection between these two programs the mobile platform is strictly kinematically uncoupled from the robot arm. This simplifies both control programs, but needs justification:

- Firstly, the project itself was divided into two halfs, the first concerned with a stationary robot with only a limited scope and the second concerned with extending that scope by introducing mobility. The consequence of this split is that at the end of the first half of the project a working control program not yet incorporating any mobile platform issues existed, and of course re-using as much parts of this program is an obvious advantage. Any tight integration of the robot arm and the mobile platform would instead have required to re-write most of the control program.

- Secondly, the kinematic coupling of a robot arm and a mobile platform is very complex. There is not yet any commercial software available for such an integration, so all systems that do exist at research facilities[6] are proprietary developments and very experimental. They imply a lot of constraints that cannot be easily ported to different hardware. The only possible solution would therefore have been to implement an in-house solution, but this was out of the scope of this work.

- Thirdly, though being a potentially interesting feature, the given task simply does not require a tight kinematic coupling. As can be seen in the task description in section 2.3 the platform and the arm only have to be used sequentially, but not simultaneously.

Because of these reasons a kinematic coupling is not considered in this work.

[6] See [Khatib et.al. 1996].

Chapter 3

Accurate Positioning of a Mobile Platform

> *"We are now cruising at a level of two to the power of twenty-five thousand to one and falling, and we will be restoring normality just as soon as we are sure what is normal anyway."*
>
> *Trillian*

One big component of the mobile robot system is the mobile platform. In general, the term "mobile platform" is a generic term attributed to almost any machine that can move autonomously on a flat surface (not including anything that walks on legs, swims or flies), and the possible incarnations are therefore far too numerous to be listed here.

In industry, for example, *autonomous guided vehicles* (AGVs) are often used for transportation tasks. AGVs can be quite large and heavy, like those in figure 3.1 built by MIAG[1]. These vehicles are used in aircraft industry for transporting large spare parts like engines or for moving an entire aircraft when the landing gear is removed[2].

Figure 3.1: Autonomous guided vehicles by MIAG used in aircraft industry.

[1] MIAG Fahrzeugbau GmbH, Kocherstraße 1, D-38120 Braunschweig, Germany. `http://www.miag.de`
[2] `http://www.miag.de/Produkte/OCS/FWW/body_fww.html`

The smaller these vehicles become, the less likely they are called AVGs, but *service robots* instead. The name suggest that these platforms are more often used for service and entertainment purposes rather than transportation tasks. The Fraunhofer IPA[3], for example, has built several platforms as in figure 3.2, which are used in the Museum for Communication in Berlin[4] as guides.

Figure 3.2: Guide and entertainment robots by the Fraunhofer IPA used in a museum.

These two examples demonstrate the immense differences in size and area of application of mobile platforms. Even though they look entirely different they basically use similar mechanisms – only at a different scale. If the application permits, they can in particular use the same control strategies and algorithms.

In this work the mobile platform is used to provide mobility to the robot arm. Simulations done by S. Plahl in [Plahl 1998] have shown that space is a strong issue in the biotechnological laboratory and the necessary placement of the devices around a stationary robot arm complicates the setup significantly. The resulting robot work cell certainly violates the project claim of allowing human personnel to use the device while the robot is idle. The basic architecture of the mobile platform used to circumvent these problems is shown in figure 3.3. This platform has motors and sensors as described in section 3.1, but – in accordance with the introduction in section 2.4 – no explicit connection to the robot arm.

[3]Fraunhofer IPA, Nobelstraße 12, D-70569 Stuttgart, Germany. `http://www.care-o-bot.de/MuseumRobots.php`

[4]Museum für Kommunikation, Leipziger Straße 16, D-10117 Berlin, Germany. `http://www.mspt.de/berlin/d211_rundgang.asp`

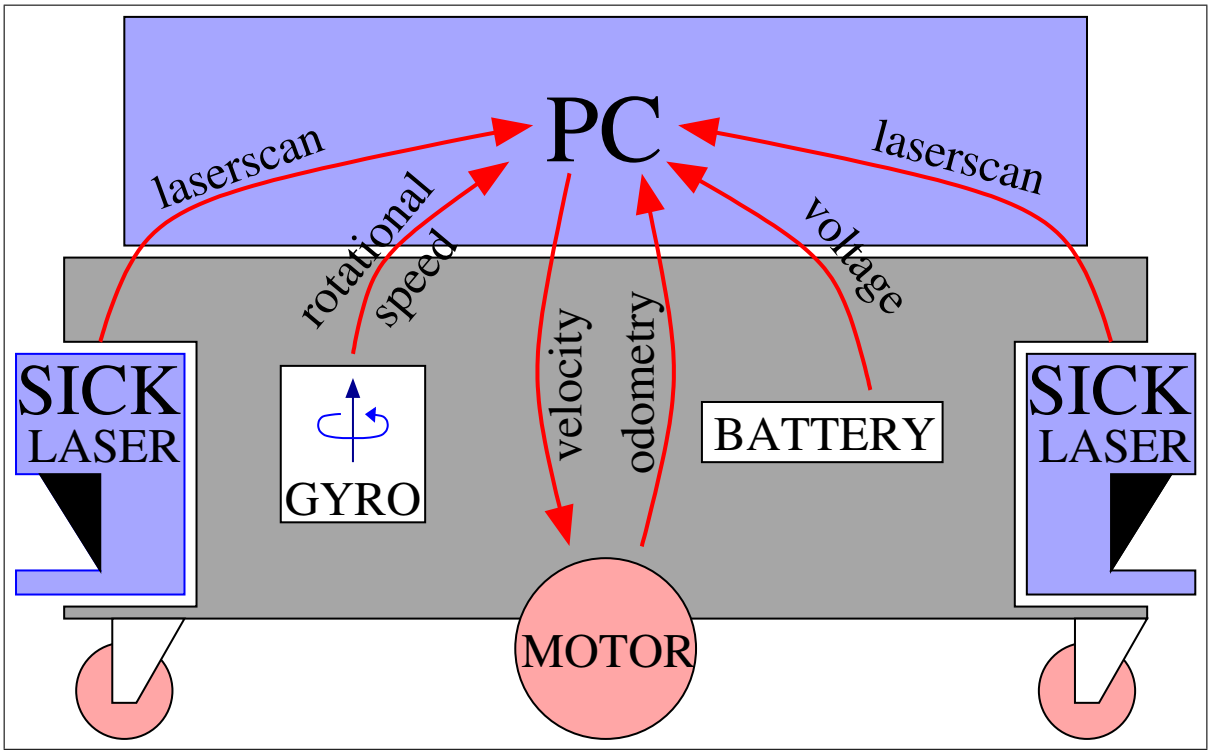


Figure 3.3: The mobile platform architecture.

With no tight kinematic coupling between mobile platform and arm the control of both components becomes much easier: The robot arm is responsible for grasping and carrying the tubes and operating the devices, and the mobile platform is responsible "only" for carrying the robot arm close enough to the devices to allow it to do that. This basically means that the platform must navigate freely in the laboratory without bumping into or otherwise damaging or harming any obstacle, including humans. It involves the tasks of *localisation*, *path-planning* and *motion execution* as defined as follows:

Localisation is the task of computing the current position of the mobile platform based on current and/or past sensor information. It is described in section 3.2. In order to give the robot arm the maximum possible operational range the positioning of the mobile platform must be of the utmost accuracy, and the accuracy of the positioning is bound to the accuracy of the localisation.

An accurate localisation is, however, not at all a trivial task because mobile platforms do usually not have sensors that deliver the *absolute* position, but only sensors that deliver – partly very noisy – *relative* measurements from which the absolute position has to be computed. An *extended kalman filter* (EKF) is therefore implemented to fuse the sensor information and compute the statistically most likeliest position. This EKF yields an accuracy that surpasses the previous commercial solutions significantly.

Path-planning is the task of computing a collision free path to a target according to a map of known obstacles. It is described in section 3.3. Different aspects of maps are discussed as well as the major path-planning issues resulting from these maps. The implemented solution uses expanded polyline maps, tangent graphs and the A^*-algorithm.

Instead of expanding each object as a single polyline it expands each line of each object separately to avoid degenerate cases. This changes the objective from "not driving into

obstacles” to “not crossing obstacle borders”, which allows motions entirely “inside” an object (for example on a table, as long as the table’s edges are not touched). The biggest part of its calculations are done off-line so that the computation time needed for a single room laboratory is neglectable.

Motion execution is the task of actually driving a path. It is described in section 3.4. Accurate motion execution in the presence of a noisy localisation is a theoretical contradiction, but the implemented solution avoids most of the common problems by a new approach. Instead of continuously and – if the localisation is too noisy – potentially endlessly tracking a target it introduces the strong notion of a desired trajectory. Using this trajectory and velocity profiles computed by a trajectory generation filter – an idea transferred from robot arms – it stops more deterministically at the target than previous approaches. The motion execution also includes a simple collision avoidance for unknown obstacles.

The remaining sections cover possible future enhancements that have not been incorporated into this work (section 3.5) and experiments and results showing the performance of the realized software (section 3.6).

3.1 The “Neobotix MP-L655“ Mobile Platform

The mobile platform is a “MP-L655” model by *Neobotix*, a division of the *Gesellschaft für Produktionssysteme* GmbH (GPS)[5]. The GPS GmbH itself is a spin-off from the Fraunhofer Institute for Manufacturing Engineering and Automation (IPA)[6]. Neobotix is a rather small company building mobile platforms and service robots as individual items on a built-to-order base, mainly for educational and scientific institutions.

The MP-L655 – for a full documentation see section A.1 in the appendices – is built as an aluminium frame with several storeys as in figure 3.4. In the basement the motors and drive and castor wheels are mounted, the first floor holds the batteries and the power electronics, the second the on-board PC and the robot arm controller and on the roof a LCD display and the robot arm itself are mounted.

The MP-L655 can be ordered with two kinematics, of which the first has been chosen for this work because a biotechnological laboratory is more likely to require complex manoeuvring than to have a very bumpy floor. The kinematics are:

1. Two spring-suspended differential drives in the rotary centre of the platform, two castor wheels at the front and one castor wheel at the back. This kinematic has the advantage that the platform can rotate on the spot. It has the disadvantage that a five-point-contact is not guaranteed to be stable on bumpy terrain. Also, the high number of castor wheels – if not properly aligned to the driving direction – introduces a lot of resistance that has to be overcome, which means that on a suboptimal floor the drive wheels are slightly more likely to slip.

[5]GPS GmbH, Department Robotics, Nobelstraße 12, D-70569 Stuttgart, Germany. `http://www.neobotix.de`

[6]Fraunhofer IPA, Nobelstraße 12, D-70569 Stuttgart, Germany. `http://www.ipa.fhg.de`

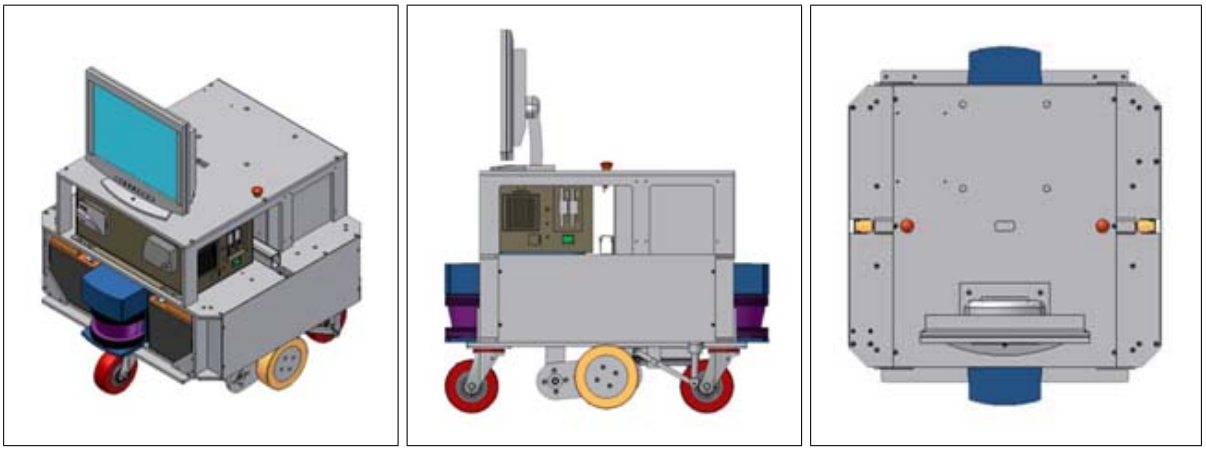


Figure 3.4: CAD drawings of the MP-L655 mobile platform (without robot arm).

2. Two differential drives at the front of the platform and one castor wheel at the back. This kinematic has the advantage that a three-point-contact is always stable and that no spring suspension is needed. It has the disadvantage that the platform cannot rotate on the spot and therefore needs more free space to perform certain manoeuvres and therefore a more complex motion planning.

Each motor has a built-in odometry sensor (described in section 3.2.1.1). In addition to these sensors the MP-L655 used in this work has an electrical gyro compass (described in section 3.2.1.2) and two SICK laser scanner (described in section 3.2.1.3), making for a total sensor set as in table 3.1. Optionally, it can be equipped with ultrasonic sensors and a bumper bar.

2 × odometry sensors (1 per motor)
1 × electrical gyro compass
2 × laser scanner (1 in front, 1 in rear)

Table 3.1: The MP-L655 sensors.

An on-board PC in a small industrial case is used to access the sensors and actors. The odometry sensors and the gyro compass are connected to this PC by a high-speed CAN (described in section A.2 in the appendices), and the laser scanner by RS422 serial lines. The entire connectivity hierarchy can be seen in figure 3.5.

At the time of purchase, the MP-L655 came with a control software called “GENCONTROL by Neobotix/GPS”. This software had several deficiencies – documented in section A.1.2 in the appendices – which made it unsuitable to be used for this work. Most notably it lacked a path planner for maze-like environments, was not accurate enough (it could arrive some 5 cm left of the target while reporting to be some 5 cm right of it) and sometimes got trapped in an endless loop driving around the target (when the catch radius around the target had been decreased in order to increase the accuracy). Because of these deficiencies the solution presented in this chapter had to be built.

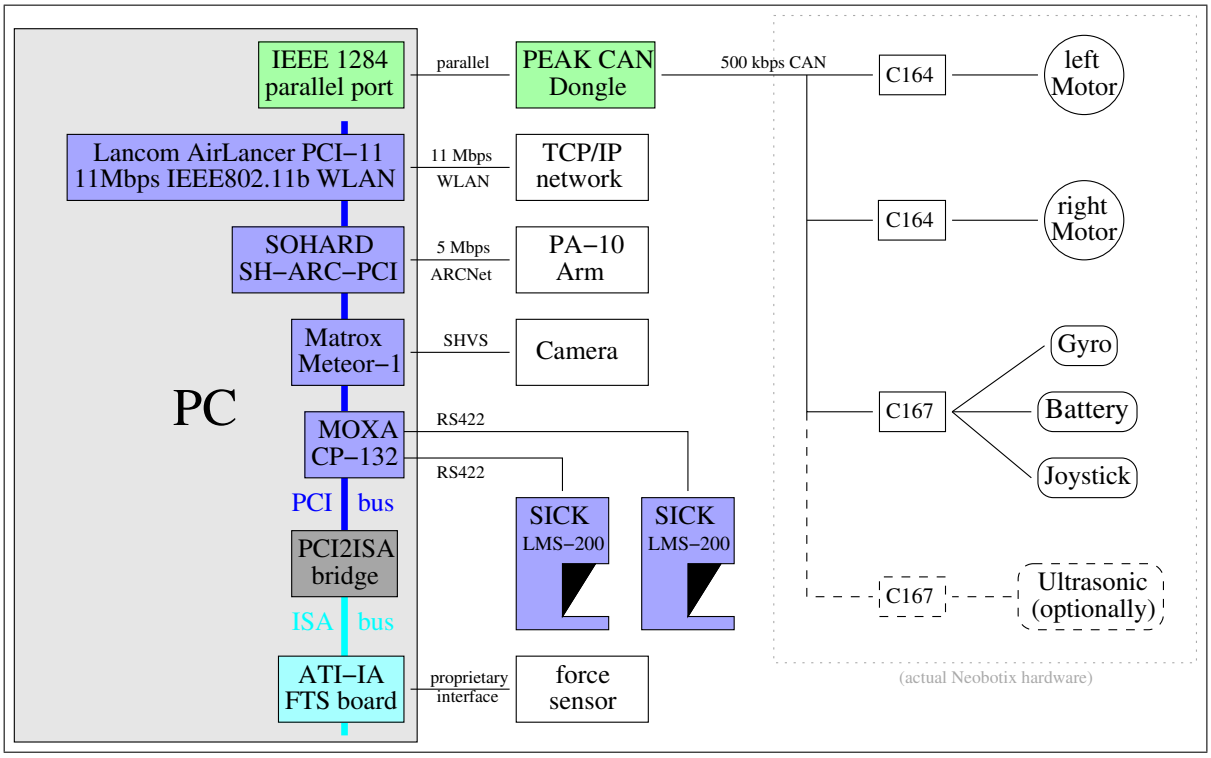


Figure 3.5: The MP-L655 connectivity hierarchy.

3.2 Localisation

As stated in the introduction to this chapter, designing a replacement for GENCONTROL involves the tasks of *localisation*, *path-planning* and *motion execution*. Localisation as the first of these tasks is the most important one because the others are based on it – an error made during localisation can not be corrected with path-planning or motion execution.

The task of the localisation is to compute the platform's position based on measurements. Any measurement is only of a limited accuracy, and therefore the position computed from them can also only be of a limited accuracy too. What that accuracy is depends on too many aspects than can be reasonably briefly listed here. Whether the achieved accuracy is satisfying depends entirely on the application.

For example, the first mobile platforms Neobotix produced were meant to be entertainment robots in the Museum for Communication in Berlin[7] (see again figure 3.2). Their purpose is to drive around in a great hall, drive up to people, welcome them, give them information about the museum and otherwise entertain them. They must stay in the hall and keep clear of all obstacles, but they are not required to physically interact with their environment. They do not need to know about positions of obstacles, only of the presence of obstacles and a strategy to avoid them. For a task like this a localisation with an accuracy of 10-20 cm is fully sufficient[8].

[7] Museum für Kommunikation, Leipziger Straße 16, D-10117 Berlin, Germany.
`http://www.mspt.de/berlin/d211_rundgang.asp`

[8] If a localisation is needed at all: Depending on auxiliary means of preventing the robots from leaving the hall it might already suffice to make them move around randomly and without knowing where they actually are.

Such an accuracy of several centimetres would already be more than enough for a human to find targets. If a human asks another human where – for example – the pipette is, he will usually get an answer like: "Oh, that's on the table where the centrifuge is." This is not an accurate answer in that it still allows for up to two square meters in which to search for a rather small pipette, but still fully sufficient. A human will just go to the table with the centrifuge (assuming he knows where that one is) and *look* for the pipette on that table. With the quality of human vision this will usually take such an infinitely small amount of time that it can hardly be called searching.

The mobile robot system in this work follows the same idea when manipulating a device: It drives to an approximate position which it assumes to be close to the device and uses its vision to look for where it really is. Since, however, its vision capabilities are much more limited than that of a human it can only search a much more limited area. More exactly, it has no strategy for searching objects by exploring a larger area than that of its initial field of view. It can explicitly not move its head and look sideways. This means that the approximate position must already be accurate enough to ensure that the object that is searched will be in the field of view of the robot's camera – a position with the quality of "on the table with the centrifuge" does not suffice.

In this section the sensors of the MP-L655 will be presented. Their integrated evaluation by an extended Kalman filter will be described. The implementation of this Kalman filter and the models it uses lead to a so far unreached localisation accuracy that safely meets the requirements of bringing the object into the robot's field of view without moving the camera.

3.2.1 Sensors

The probably most commonly known method to measure a position is the *global positioning system* (GPS), which computes a position based on the measurement of distances to satellites orbiting the earth on known paths. Although using GPS sounds like a promising idea at first sight, it is not.

One problem is that GPS computes a position only, not an orientation. In order to obtain an orientation at least two independent receivers are needed. From the different position measurements of these receivers the orientation of whatever object they are mounted on can in theory be computed. In practice this is unlikely to work for small objects like mobile robots because there is yet another problem.

The second problem is that the resolution of GPS is too bad. Normal GPS has a resolution of about 20 m without the intentional disturbance called *selective availability* (SA) that was put on the signal by the US military until may 2000 (before that it was more like 200 m for civilian use[9]). This is because the measurement of distances to satellites depends – amongst other things – on the number of visible satellites and the atmospheric distortions.

Differential GPS (DGPS) can be used to improve this situation. DGPS uses a fixed GPS receiver whose position has been determined by other means with more accuracy. This receiver can then compute the error between its real position and what GPS reports as its position and forward it to other clients close by, which are supposed to suffer from the same distortions. The client receivers can then compensate this error, resulting in a remaining error as low as

[9] `http://www.igeb.gov/sa`

1 m. Depending on the range of these system they are also called *local* or *wide area augmentation systems* (LAAS/WAAS). A LAAS can for example be used at airports to provide a better navigation assistance for planes in the range of up to 50 km distance[10].

Even though this is a considerable improvement compared to standard GPS it is still not accurate enough for this work. Radio based systems with an accuracy in the range of what this work needs – about 1 cm – have not been found commercially available, and even if they were it would still have to be verified that they work reliably in a building consisting of ferro-concrete and with lots of metal objects. For most mobile platforms other sensors are therefore used.

3.2.1.1 Odometry

Odometry sensors are sensors which compute a position by measuring and integrating over the travelled distance (greek: *hodometron*, *hodos* = "way" and *metron* = "measure"). This does safely yield an absolute position of the device they are connected to, but in some cases this position is not the ultimately desired information.

In case of a standard robot arm odometry sensors are attached to the drive motors as in figure 3.6. These *encoders* mostly count axis revolutions only, but given an initial calibration value the axis orientation can be computed by integrating the revolutions over time. This computation is guaranteed to be stable and accurate because due to the fixed kinematic coupling no slippage between the drive motor and the next arm segment can occur.

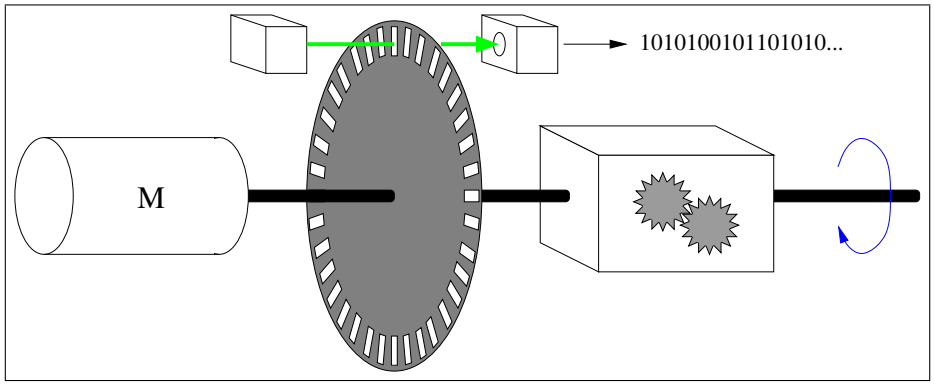


Figure 3.6: Schematic construction of odometry encoders.

In case of a mobile platform the situation is different. Odometry sensors can still provide an absolute orientation of the wheels, but that is not the desired information. The desired information is the position and orientation of the platform, and that can only be computed from the odometry by a function called *kinematics* that is not unique and/or not invertible (see section 3.2.3 for details).

Because of this, odometry – in the context of a mobile platform – can not be considered an *absolute* measurement. It only *tracks* the position of the platform by integrating over the wheel revolutions, having no idea what the real absolute platform position is. Yet, it is common to use odometry because its short-term accuracy – for example for fine-positioning – is unchallenged.

[10] `http://gps.faa.gov/Programs/LAAS/laas.htm`

$c_{\text{ticksPerMotRev}}$	=	4096
$c_{\text{gearRatio}}$	=	37
t_{sample}	=	0.002 s

Table 3.2: System parameters of the motors and wheel encoders.

The MP-L655 has wheel encoders in both of the two drive wheels with parameters as in table 3.2. The encoder resolution of 4096 ticks per motor revolution in combination with the gear ratio of 1:37 means that this mechanism has a drive wheel revolution resolution of 0.0024 ° as per

$$p_{\text{deg}} = \frac{p_{\text{enc}} \cdot 360^\circ}{c_{\text{ticksPerMotRev}} \cdot c_{\text{gearRatio}}}$$

where

$$\begin{aligned} [p_{\text{deg}}] &= {}^\circ\,(\text{degree}) \\ [p_{\text{enc}}] &= 1\,(\text{ticks}) \end{aligned}$$

Velocities are not handled in degrees per second, but in ticks per sample interval of 0.002 s. The necessary conversion is

$$v_{\text{enc}} = \frac{v_{\text{deg}} \cdot c_{\text{ticksPerMotRev}} \cdot c_{\text{gearRatio}} \cdot t_{\text{sample}}}{360^\circ} \tag{3.1}$$

where

$$\begin{aligned} [v_{\text{enc}}] &= 1\,(\text{ticks/sampleinterval}) \\ [v_{\text{deg}}] &= {}^\circ/s \end{aligned}$$

The wheel encoders are tightly integrated with the motor and can be read out by accessing the appropriate controller on the CAN.

3.2.1.2 Gyro Compass

Compasses (ital. *compassare* = to pace off) are devices to indicate a certain fixed point of the compass and are used to determine orientations. Classic magnetic compasses using the magnetic field of the earth are easily disturbed by the presence of metal or electricity and therefore not suitable for precise and reliable measurements. Air planes for example use gyroscopes instead.

A gyroscope is a device which takes advantage of the physical law of the conservation of angular momentum. It contains a freely suspended mass that – once brought into rotation – will keep its absolute orientation even if the outside world rotates (as seen from its point of view). The only two disadvantages are that a gyroscope is a delicate and complicate mechanical construction and that depending on the quality it is made with suffers from a long-term drift. The latter is similar to the odometry because it does not measure an absolute value, but only integrates over changes with no compensation for any errors. In the aviation technology they can still be used because it is sufficient to re-adjust them once every few hours when other (absolute) measurements are available. In case of the mobile platform these absolute measurements are given by the laser scanners described in section 3.2.1.3.

The MP-L655 is equipped with a *BEI* [11] "GYROCHIP® Model HORIZON" electrical gyro as in figure 3.7. This gyro uses a miniaturised oscillating quartz tuning fork as sensitive element. When brought into rotation a coriolis force is exerted on the fork, leading to a change of the electric charge in it that can be measured. The gyro then outputs a voltage that is linearly correlated to an angular velocity of up to ± 90 °/s.

item	value
Size (L×W×H):	58.25 × 25.25 × 25.25 mm
Input voltage:	+8 to +15 V DC
Output range:	+0.5 to +4.5 V DC
Standard range:	± 90 °/s
Operating temperature:	-25 to +70 °C
Shock:	200 g
Short term stability:	<0.05 °/s

Figure 3.7: The GYROCHIP® Model HORIZON electrical gyro with its technical specification.

On the MP-L655 the gyro is connected to the C167 controller on the I/O board. The angular velocity reading is digitised by the controller, integrated and can be read out by accessing it on the CAN. Although this yields – once initialised – an absolute orientation, this value is again differentiated to an angular velocity by the localisation. This is because it is much safer to compensate small relative errors in each cycle than to allow a larger absolute error to accumulate in the first place. This way problems with the gyro drift can be avoided before they arise.

Examples of this drift can be seen in figure 3.8, which shows results from three measurements of almost 1 h. The experiments were done consecutively and under apparently the same conditions. A comparison with a constant drift of -0.005 °/s shows that all three runs are by more than a factor of 10 under the specified short term stability of 0.05 °/s. The fact that the runs were done consecutively suggests a systematic reason for their different appearance, most probably the room temperature affecting the A/D converter or similar effects. The possibility of such an uncontrollable dependence emphasises that the absolute value of the gyro should not be relied on.

3.2.1.3 Laser Scanners

As has been mentioned in the two previous sections both the odometry and the gyro compass suffer from drift effects as they only integrate changes with limited resolution and do not have a reference to an absolute frame. In order for this drift to not have a fatal effect on the localisation an additional sensor is therefore needed, a sensor which does have a reference to an absolute frame. On the MP-L655 two laser scanners are used for this task.

The *SICK*[12] "LMS-200 Laser Measurement System" (laser scanner) in figure 3.9 is a non-contact measurement system that scans a 2d-plane of its environment. It works on the general

[11] BEI Technologies Inc., Systron Donner Inertial Division, 2700 Systron Drive, Concord, CA 94518-1399, USA. `http://www.systron.com`

[12] SICK AG, Auto Ident Devision, Nimburger Str. 11, D-79276 Reute, Germany, `http://www.sick.de`

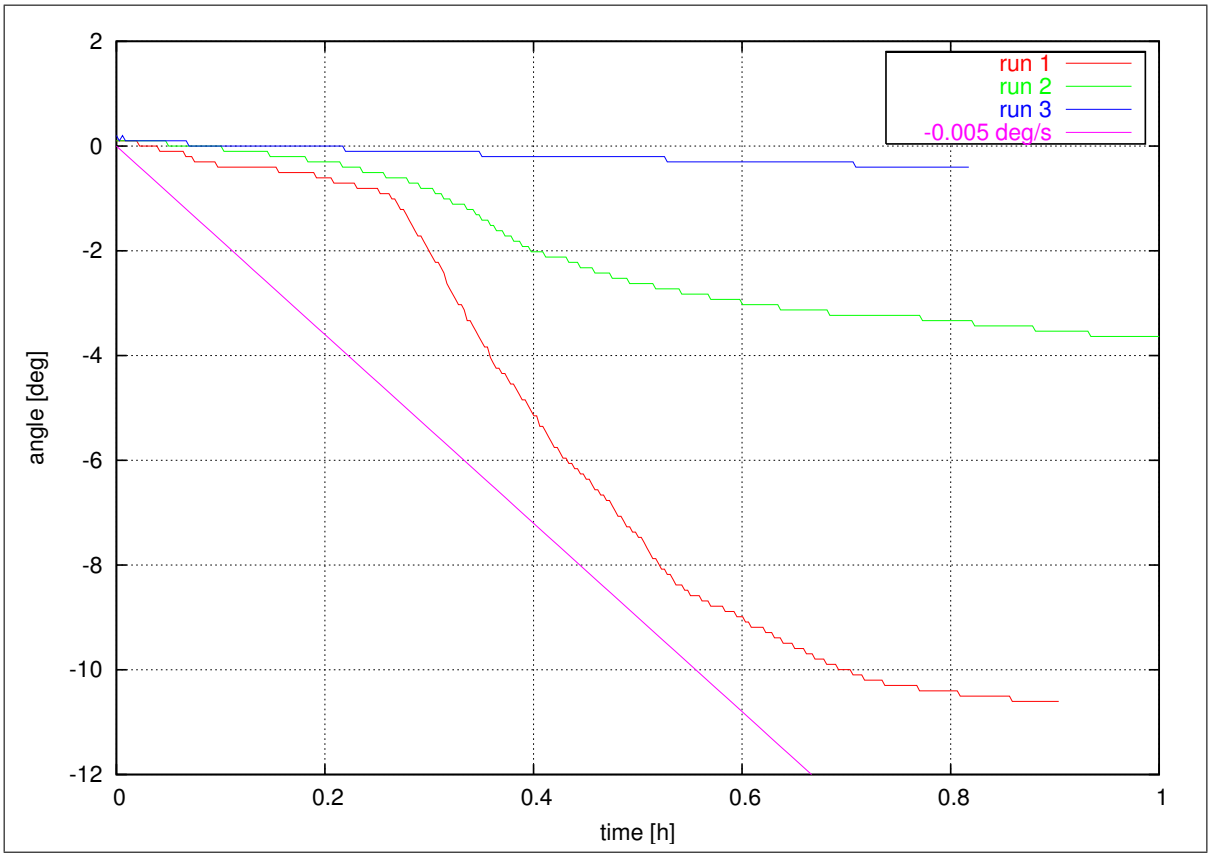


Figure 3.8: Three runs of measuring the drift of the gyro compass over about 1 h each at a completely immobile platform. The line indicating -0.005 °/s is only 1/10th of the specified short term stability.

principle of measuring the time of flight of a laser beam. As figure 3.10 illustrates, a laser light source S emits short light pulses, which are then reflected by objects and thrown back to the scanner where they are detected by a receiver R. The time Δt between emitting a light pulse and receiving its reflection is proportional to the distance s between the object and the scanner. The LMS-200 extends this principle by using a rotating mirror to redirect the laser beam to not only cover one direction, but a complete semi-circle in front of it.

Size (H×W×D):	185 × 155 × 156 mm
Weight:	4.5 kg
Operating power:	24 V, ≤ 20 W
Operating temperature:	0..+50°C
Operating range:	about 8 m with mm resolution, up to 80 m with cm resolution
Angular resolution:	1, 0.5 or 0.25 °
Systematic error (8 m range):	typ. ±15 mm
Statistical error (8 m range):	typ. 5 mm

Figure 3.9: The SICK LMS-200 laser scanner.

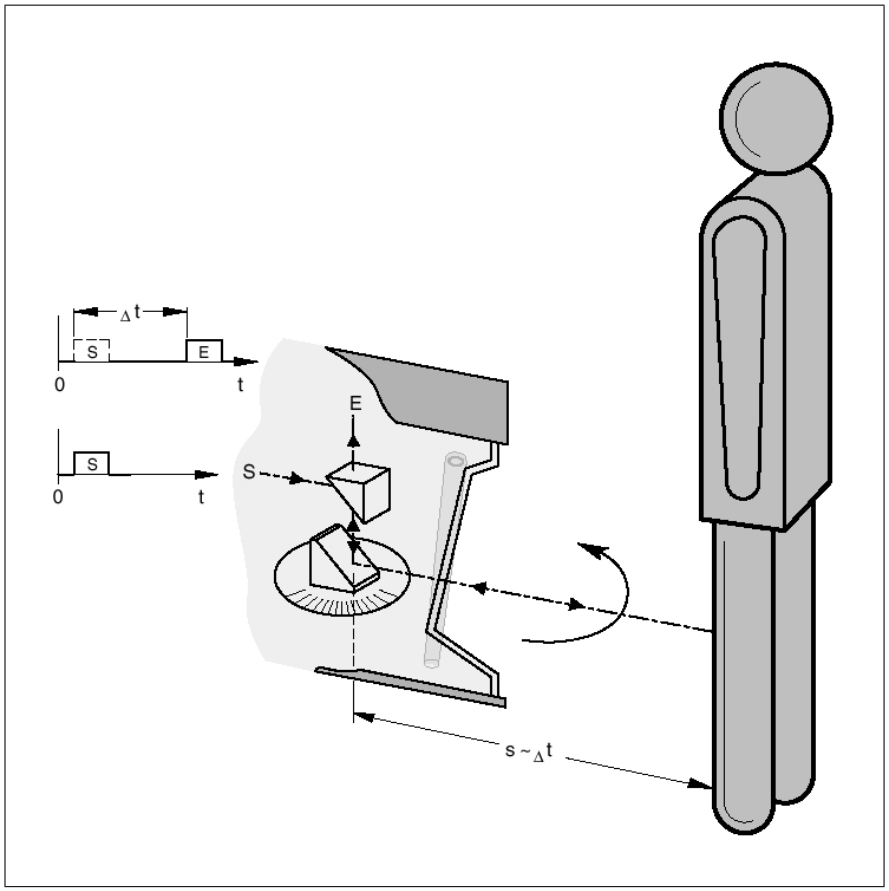


Figure 3.10: The basic principle of operation of the LMS-200 laser scanner.

In addition to the time of flight Δt the LMS-200 can also measure the intensity of reflectance (the ratio I_R/I_S). This way special reflector marks (called *beacons* or *landmarks*)[13] can be distinguished from other objects. These beacons can be used by the localisation to compute the platform position by triangulation, the details of which will be described in the next sections. To illustrate this, figure 3.11 shows a sample scan of an almost empty room.

Caution has to be taken if two or more laser scanners are to be operated with intersected fields of view. Although it is possible to electrically synchronise two scanners so that their mirrors rotate in a way that this problem does not occur, SICK recommends mounting them in a way that their planes of view do not intersect in the first place. This presents a problem for multiple mobile platforms in a single room as synchronisation is not possible and tilting the scanners is not feasible. Since in this work only one platform is used this problem is stated here, but not further investigated.

According to [SICK AG 2002], the LMS-200 laser scanners are *"not devices for protecting persons as defined by current machine safety standards"*. On the MP-L655 platform, however, they *are* used for localisation as well as collision avoidance, so for a full safety license the platform would need additional bumpers and/or other proximity/collision sensors. Currently SICK is introducing a new model, the S3000, which will comply to safety regulations, but it has not been evaluated for localisation purposes. Again, since these aspects do not affect this work, they are not further investigated here.

[13] Fa. IMOS Gubela GmbH, Postfach 1113, D-77867 Renchen, Germany. `http://www.imos-gubela.de`.

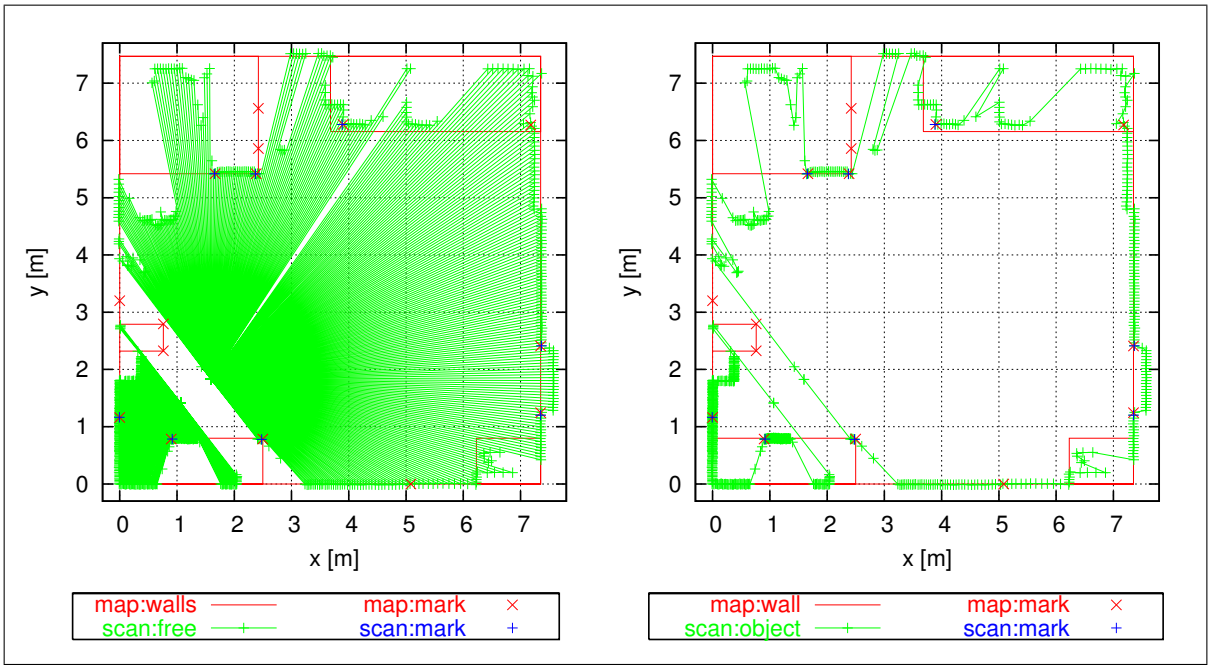


Figure 3.11: Sample scan of an almost empty room as seen from a mobile platform close to the lower left corner. On the left rays have been drawn from the scanners to the scan points to indicate the free space, whereas on the right the scan points have been connected with a line to indicate the object(s) outline. The reflector marks from the map and as seen by the scanner are also shown. The empty strip through the platform is its blind area and caused by the way the scanners are mounted.

The LMS-200 laser scanners are connected to the PC using a serial interface which can be configured as either EIA-RS232 or EIA-RS422. On the MP-L655 they are operated in RS422 mode because this mode is considered to be more robust against electrical interference. Over the serial line the scanners can be sent commands according to the *Telegram Definition Manual* (TDM) [SICK AG 2000], which they execute autonomously using a built-in processor. The communication details and protocol are documented in the appendices in section A.3.

3.2.2 Kinematics Issues

Robotics is (mostly) about moving, and moving means to change some *degrees of freedom* (DOFs) of something with respect to some reference coordinate system, or, in other words, to change its *position* and/or *orientation*. Usually the DOFs can only be modified indirectly by the actors of a robot mechanism and so a mapping $\theta_i \rightarrow p_i$ between the parameter θ_i the actor i controls and the DOF p_i that is actually desired to be controlled must be established. This mapping is called the *forward kinematics* of the mechanism and is straightforward for translational and rotational joints:

- Computing $\theta_i \rightarrow p_i$ as the position $p_i = x$, y or z of a translational joint i in a *cartesian robot* is trivial because the actor and the DOF are fixedly linked by a single worm gear or belt.
- Computing $\theta_i \rightarrow \alpha_i$ as the orientation $p_i = \alpha_i$ of a rotational joint in a *single segment arm* is trivial because the actor and the DOF are fixedly linked by a gear box.

- Computing $\vec{\theta} \rightarrow \vec{p}$ as the cartesian position $\vec{p} = (x, y, a, \alpha_{\text{roll}}, \alpha_{\text{pitch}}, \alpha_{\text{yaw}})$ of the end-effector of a *multi-segment arm* is slightly more complex because the effects of the joints are overlayed, but can still be easily established.

The forward kinematics is often *unique*, meaning that one set $\vec{\theta}$ of actor values corresponds to exactly one set $\vec{p}$ of DOF values only. For example, when driving individual joints of a multi-segment arm one after another the sequence does not matter. In case of a 2-segment arm the different chains $\Delta\theta_1$ and $\Delta\theta_2$ yield two positions

$$\begin{array}{rcl} (\Delta\theta_1, 0) + (0, \Delta\theta_2) = (\Delta\theta_1, \Delta\theta_2) & \Rightarrow & \vec{p}_1 \\ (0, \Delta\theta_2) + (\Delta\theta_1, 0) = (\Delta\theta_1, \Delta\theta_2) & \Rightarrow & \vec{p}_2 \end{array} \tag{3.2}$$

but it trivially follows that

$$\vec{p}_1 \quad = \quad \vec{p}_2$$

The reason is that even though the mapping is called "kinematic" it only addresses the *static* part of the physics, i.e. the *position*, but not the *motion*. The resulting position, however, is not affected by the order of application of changes to individual θ_i for these type of kinematics.

For a mobile platform this is no longer true. The reason is that the underlying mapping $\vec{\theta} \rightarrow \vec{p}$ is no longer unique. Taking the sequence of changes from equation 3.2 it has to be stated that

$$\vec{p}_1 \quad \neq \quad \vec{p}_2$$

even though the actor values $(\Delta\theta_1, \Delta\theta_2)$ are the same in both cases.

As figure 3.12 illustrates a mobile platform with the MP-L655 kinematic will turn to the side if only one motor is moved. If, for example, first only the right and then only the left motor is used for the same amount of time the platform moves towards the red end position. In case of the opposite sequence it moves towards the green end position. If the motors had been moved simultaneously the platform would have moved straight forward. Therefore the resulting position depends on which motor is moved first, meaning that the forward kinematic is not unique. As a consequence the whole problem of "forward kinematic" has to be addressed differently for a mobile platform than for a robot arm.

3.2.3 Dead Reckoning

One way of addressing the forward kinematics problem is to use *dead reckoning* (DR). DR is the general idea to compute a state by continuously tracking (integrating) changes (differentiations) of that state instead of computing it directly from single measurements. One reason for doing this is that it is often easier to measure travelled distances than an absolute position, and another reason is that for a mobile platform with just odometry sensors it is actually the only way to implement a forward kinematic. One set of wheel angles (θ_L, θ_R) can correspond to many positions (x, y, α), and only a continuous history of the motions can single out the correct one.

DR can – depending on the sensors used – be very accurate over "short" distances, but it has no bounds for the error on "larger" ones. One reason for this is that the state changes are often measured with limited resolution, and another reason is that in general all measurements are afflicted with noise. Integrating over these errors means that the resulting value will become

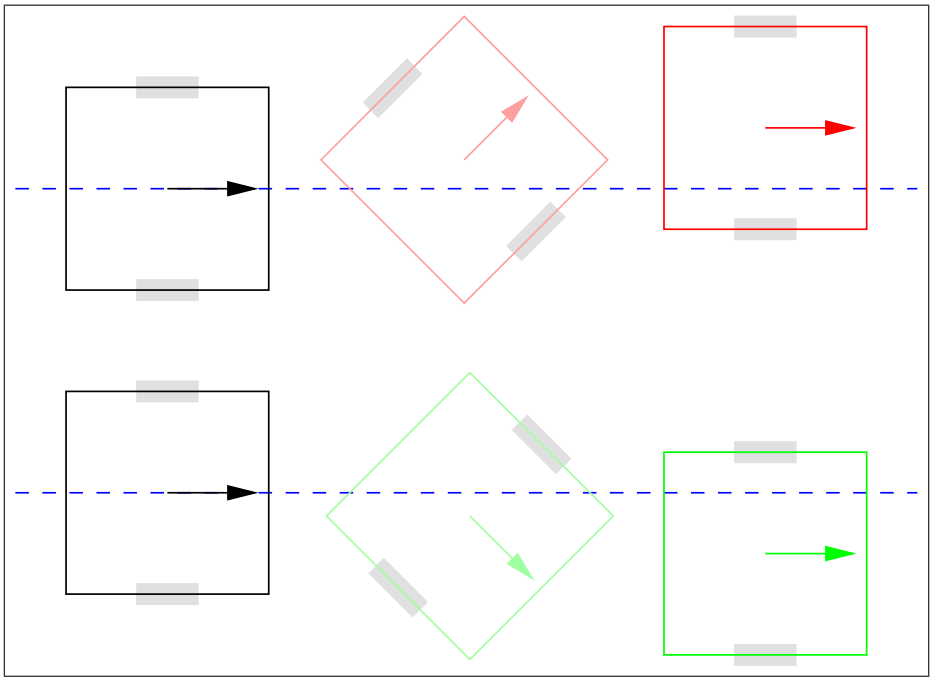


Figure 3.12: Different sequences of changes of actor positions lead to different world positions for a mobile platform.

more and more inaccurate over time. Therefore, if the state computed by DR is not checked against other, absolute measurements from time to time, it may become arbitrarily wrong.

The practical upshot of this is that DR must not be the only localisation method used on a mobile platform, but still it is always part of the localisation because it is very accurate over short distances. The quality of DR depends on the kinematic model and the accuracy of the measuring equipment.

DR is not at all a new idea, but in fact quite old in human history. Amongst the most famous people to have used DR on a large scale "motion" is Christopher Columbus with his 1492 voyage to "India"[14]. He navigated by reading the direction from a compass (however accurate that may have been) and measuring the speed by checking the flow of water close to the ship. Whether he was sailing with or against an ocean current he did not know, and therefore this measurements were inaccurate. And he did, after all, arrive quite wrong.

3.2.3.1 Improving the Kinematic Model

The original GENCONTROL kinematic model was based on the fact that the dynamic state of the platform consisted of only an angular velocity ω (about the platform's vertical axis) and a translational velocity v (along whatever direction the platform is currently heading to). More specifically, since the localisation is a discretised process these values were expressed as differential values $\Delta\alpha$ and Δs for some time interval Δt. The update step for a Δt in the localisation was:

$$x(t+1) = x(t) + \Delta s \cdot \cos(\alpha) \quad (3.3)$$

$$y(t+1) = y(t) + \Delta s \cdot \sin(\alpha) \quad (3.4)$$

$$\alpha(t+1) = \alpha(t) + \Delta\alpha \quad (3.5)$$

[14] See `http://www1.minn.net/~keithp/index.htm`.

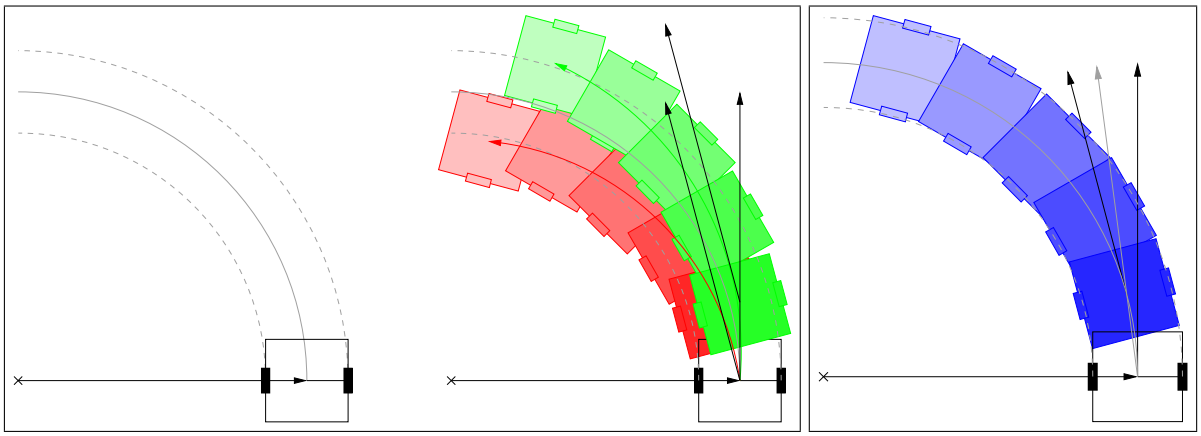
with

$$\begin{aligned} \Delta\alpha &= \frac{(\Delta\alpha_R - \Delta\alpha_L)\cdot r}{2d} \\ \Delta s &= \frac{(\Delta\alpha_R + \Delta\alpha_L)\cdot r}{2} \end{aligned} \tag{3.6}$$

being computed from the drive wheel angular changes $\Delta\alpha_L$ and $\Delta\alpha_R$, the radius r of the wheels and their distance d to the centre of the platform. At first sight this seems to fit to the kinematics of the MP-L655, which is described as to be able to

- translate forward and backward and
- rotate on the spot,

plus superimpositions of these. Only a second sight reveals that the superimpositions present a problem: All possible motions are rotations about some point on the line formed by the two drive wheels, say, a partial circular arc. But with the above equations it is not possible to track the position of such a motion. Figure 3.13(a) shows that the computed position does not stay on the real trajectory. The red sequence occurs when first the orientation is applied and then the translation, and the green sequence occurs when the order of application is reversed.



(a) Original GENCONTROL kinematic. (b) Improved kinematic.

Figure 3.13: Positions as computed by DR using different kinematics when driving a circular arc.

This is because the translational component Δs is always tangential to the circular arc. If one first updates the position and then the orientation the platform will end up outside the arc, and if one first updates the orientation and then the position it will end up inside the arc.

In both cases the computed change of orientation $\Delta\alpha$ would have to be modified to put an emphasis on getting back on the arc, and that GENCONTROL does not do.

A first improvement is to compute the update as:

$$\begin{aligned} \beta &= \alpha(t) + \frac{\Delta\alpha}{2} \\ x(t+1) &= x(t) + \Delta s \cdot \cos(\beta) \qquad (3.7) \\ y(t+1) &= y(t) + \Delta s \cdot \sin(\beta) \qquad (3.8) \\ \alpha(t+1) &= \alpha(t) + \Delta\alpha \end{aligned}$$

Using these equations the position is always held on the arc, regardless of what the current orientation α or the translational speed Δs might be (see figure 3.13(b)).

Another problem is that Δs as per equation 3.6 is only correct for $\Delta\alpha_R = \Delta\alpha_L$ (translation straight ahead) or $\Delta\alpha_R = -\Delta\alpha_L$ (rotation on the spot). This is because it computes the length of the circular arc of the centre of the platform, not the length of the circular chord. Focusing on the arc is understandable because the arc is the trajectory the centre of the platform has moved, but it is the chord that is used by the equations 3.7 and 3.8. Since – except from the two above mentioned cases – the length of the arc is always greater than the length of the chord these equations yield a position which is almost always too far ahead. The argumentation is that since an only small error is introduced by this inaccuracy it is non-fatal because the dead-reckoning is fused with other localisation techniques which will correct this error, but still this model can be improved.

What is needed for a correct computation of Δs is the length of the circular chord. Based on the general formula for a circular arc $s_{\text{arc}} = r \cdot \alpha$ where α is already known from above it can be established that

$$r = \frac{s_{\text{arc}}}{\alpha}$$

in which of course $r \to \pm\infty$ for $|\alpha| \to 0$ because there is no "radius" if the platform is moving straight ahead. At the same time the length s_{chord} of a circular chord can be computed as

$$s_{\text{chord}} = 2r \sin\frac{\alpha}{2}$$

Combined, this yields

$$s_{\text{chord}} = s_{\text{arc}} \cdot \underbrace{\frac{2}{\alpha}\sin\frac{\alpha}{2}}_{f(\alpha)} \tag{3.9}$$

where $f(\alpha)$ can be considered a "correction factor" for s_{arc}. This factor has to be analysed to be sure the division by α is non-fatal. Together with $\sin z = \sum_{k=0}^{\infty}(-1)^{k+2} \cdot \frac{z^{2k+1}}{(2k+1)!}$ for complex arguments z it follows that

$$\begin{aligned} f(\alpha) &= \frac{2}{\alpha}\sin\frac{\alpha}{2} \\ &= \frac{2}{\alpha}\left(\sum_{k=0}^{\infty}(-1)^{k+2} \cdot \frac{\left(\frac{\alpha}{2}\right)^{2k+1}}{(2k+1)!}\right) \\ &= \sum_{k=0}^{\infty}(-1)^{k+2} \cdot \frac{\frac{2}{\alpha} \cdot \left(\frac{\alpha}{2}\right)^{2k+1}}{(2k+1)!} \\ &= \sum_{k=0}^{\infty}(-1)^{k+2} \cdot \frac{\left(\frac{\alpha}{2}\right)^{2k}}{(2k+1)!} \end{aligned}$$

The division by α disappears and $f(\alpha)$ is therefore safe to be used for values of α near to 0. As can be seen in figure 3.14 this factor is indeed always < 1 except for $\alpha = 0$. Using this factor the true chord length s_{chord} can be computed as per equation 3.9 and used in equations 3.7 and 3.8 to finally yield "correct" results.

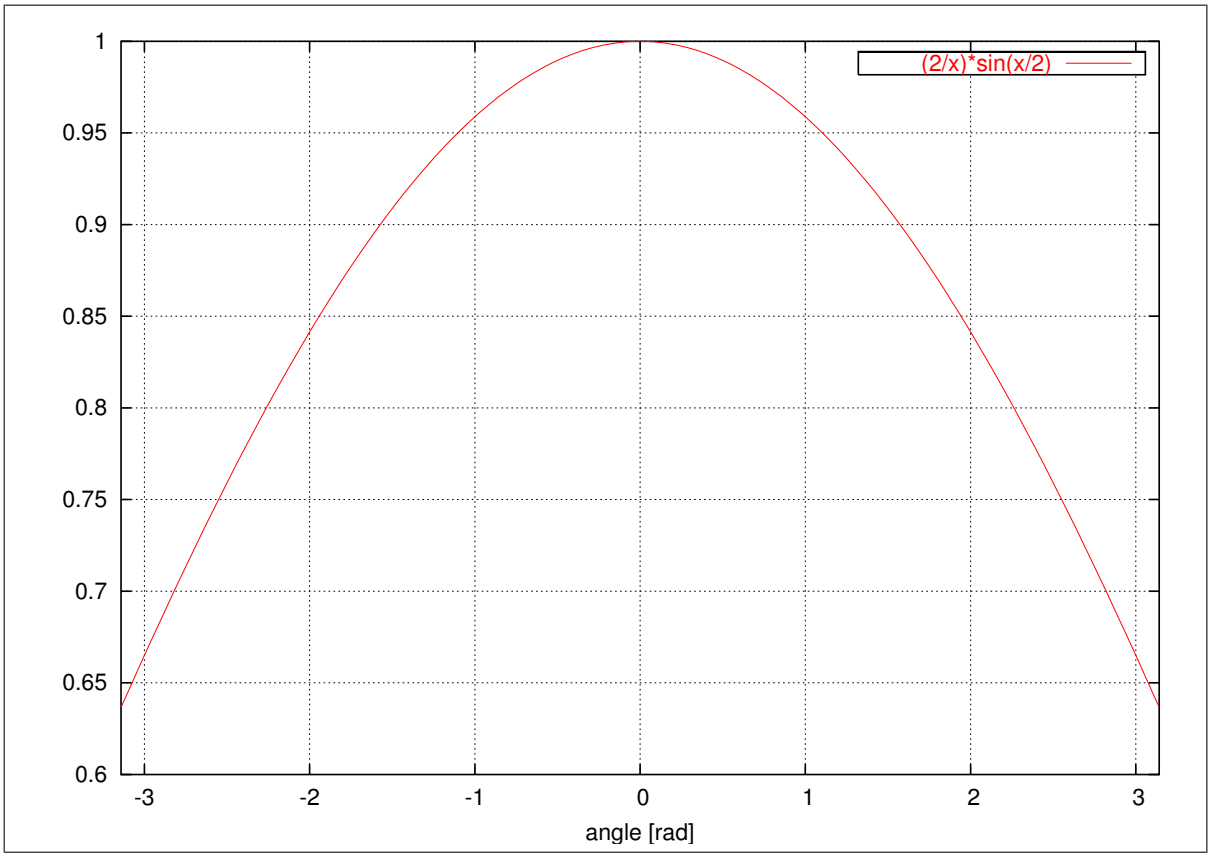


Figure 3.14: The correction factor $f(\alpha)$ used to compute the circular chord length given a circular arc length.

Note that the results are still not absolutely "correct" because one link is still missing. The $\Delta\alpha$ used by the above equations actually has to be called $\Delta\alpha_{\text{odo}}$ because it only takes the odometry into account. The gyro compass, which computes its own $\Delta\alpha_{\text{gyro}}$ is not considered. Integrating it by other means than building a – perhaps weighed – average of $\Delta\alpha_{\text{odo}}$ and $\Delta\alpha_{\text{gyro}}$ is not yet possible because so far no model exists that describes how the two measurements correlate. Only the Kalman filter in section 3.2.5 will introduce such a model.

3.2.3.2 Borenstein Tests

Johann Borenstein has done extensive studies about sensors and methods for mobile robot positioning. His work "Where am I?" [Borenstein, Everett and Feng 1996] is regarded a standard book on mobile robotics. He has also given name to a benchmark for measuring odometry errors in mobile robots [Borenstein and Feng 1995]. This "Borenstein Test" tries to provoke both systematical errors caused by a faulty kinematics and non-systematical errors caused by slipping and other problems with the floor. Its aim is to allow the user of a mobile platform to make the odometry as accurate as possible so that the platform can cover larger distance and needs fewer resource-consuming absolute measurements.

It has to be kept in mind that the Borenstein test is from 1995 when computers were by a factor of probably around 50 slower than today (2004), so at least the argument of available resources for computation of absolute measurements is no longer that urgent. With a measurement rate of the laser scanners of 38 Hz and a maximum (translational) speed of the MP-L655 of 1 m/s

the platform will only have moved at most 2.6 cm between two laser measurements, and over such a short distance odometry errors are very unlikely to accumulate. More precisely, since the path controller and the odometry are in the current configuration run with only 31.25 Hz it is actually impossible. Therefore no true Borenstein tests have been done with the MP-L655.

To demonstrate the possible magnitudes of the error of the odometry a simplified test has been done instead. In this test the mobile platform is commanded to drive a square with a side length of 2 m using its full Kalman filter localisation including absolute landmarks. Parallel to that, a second Kalman filter instance is run with only the odometry update. By using only the wheel encoders and the gyro compass but with the Kalman filter model of the system the test is close, but not identical to the original Borenstein test.

As can be seen in figure 3.15 the “desired” (true) trajectory forms the intended square, but the “real” (odometry) trajectory deviates very quickly. It actually deviates the more the lesser the base velocity is, which indicates that not the travelled distance is the primary problem, but the time the platform needs for that distance: If the platform moves slower it needs more localisation cycles for the same distance and therefore a larger error can accumulate. The fact that all deviations tend to go in the same direction indicates a systematic problem. In general, the test shows that odometry can not be trusted over even short distances, let alone if the floor is slippery.

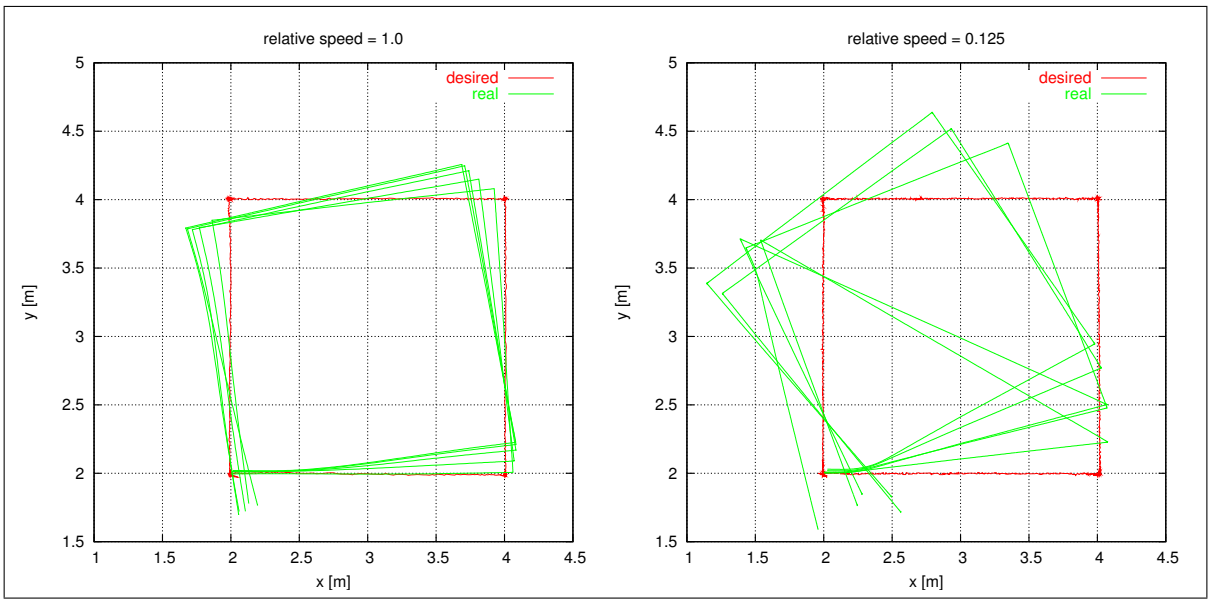


Figure 3.15: The mobile platform driving a square with a side length of 2 m with two different velocities. Each test is repeated five times.

Borenstein has also introduced the *gyrodometry* idea to combine measurements from a gyro with measurements from wheel encoders. In [Borenstein and Feng 1996] he writes:

> *Sensor-fusion of this kind has been done before, usually by means of a statistical model that describes the behaviour of the gyro and the behaviour of the odometry component. However, because these systems are based on models, they can not anticipate the unpredictable and potentially ”catastrophic” effect of larger bumps or objects occasionally encountered on the floor. By contrast Gyrodometry has been*

> *developed based on a careful study of the physical interaction between the ground and the vehicle. We have found experimental evidence that non-systematic odometry error sources (such as bumps) impact the vehicle only during very short periods; typically a fraction of a second for each encounter. During these short instances the readings from the gyro and from odometry differ significantly, while in the absence of large non-systematic errors the readings are very similar. Gyrodometry makes use of this observation by using odometry data only – most of the time, while substituting gyro data only during those brief instances during which gyro and odometry data differ substantially. This way the ill-effects of gyro drift are almost completely eliminated, and our method can thus make use of inexpensive gyros with large drift rates.*

The idea of gyrodometry is only presented as a possible enhancement here. It has not yet been applied to the MP-L655 platform.

3.2.3.3 Summary

Taking a wheel encoder resolution of 0.0024° into account DR is the method of choice to provide a high accuracy for small incremental motions. However, it has to be calibrated against absolute measurements in regular intervals because due to several inaccuracies it can arbitrarily drift off the real values over time. It also requires an *initial value* because it can only track changes of that value, not the value itself.

3.2.4 Least Squares Fit

In order to compensate the drift problems of DR an absolute reference is needed. For the MP-L655 platform used in this work special reflector marks for the laser scanners are therefore used as absolute references. These *landmarks* are placed throughout the laboratory at fixed positions and their coordinates are stored in a map. By referring to these known landmarks an absolute position can be computed by triangulation. This process of triangulation is not trivial and will be discussed in this section.

3.2.4.1 Initial Guess

The first step in the process is to safely identify the landmarks. Although the final goal is an *absolute* position these scanners still provide a *relative* measurement, i.e. they yield a set of tuples

$$(d_1, \alpha_1), (d_2, \alpha_2), ..., (d_n, \alpha_n)$$

of distances d_i and angles α_i under which reflector marks are seen in the platform coordinate frame. Using these values and given an *initial guess* of the platform position $({}^p\tilde{x}, {}^p\tilde{y}, {}^p\tilde{\alpha})$ the coordinates $({}^m\tilde{x}_i, {}^m\tilde{y}_i)$ of a mark as seen by the platform can be given as

$$\begin{aligned} {}^m\tilde{x}_i &= {}^p\tilde{x} + d_i \cdot \sin({}^p\tilde{\alpha} + \alpha_i) \\ {}^m\tilde{y}_i &= {}^p\tilde{y} + d_i \cdot \cos({}^p\tilde{\alpha} + \alpha_i) \end{aligned}$$

These "predicted" coordinates can be used to establish a mapping $({}^m\tilde{x}_i, {}^m\tilde{y}_i) \rightarrow ({}^m x_i, {}^m y_i)$ of a mark as seen by the platform onto a mark as stored in the map. This mapping may not be possible for all marks because some of them may only be false reflections by highly reflecting objects like glass or metal and have no corresponding marks in the map. As a result the mapping

process must take care to only map those seen marks on a map mark that actually do have a map mark close by, and at most only one seen mark per map mark.

The existence and quality of the initial guess it the crucial point in this process, as is illustrated in figure 3.16. The marks in the upper part of the figure are printed in different colours and therefore the situation is unique. The marks in the lower part of the figure are not distinguishable, and therefore – if it was not for the square frame – the three situations can not not be told apart. A laser scanner certainly can not tell them apart because without additional information the three situations are entirely symmetric. It is because of these symmetries that an initial guess is needed to settle on one of the possible solutions.

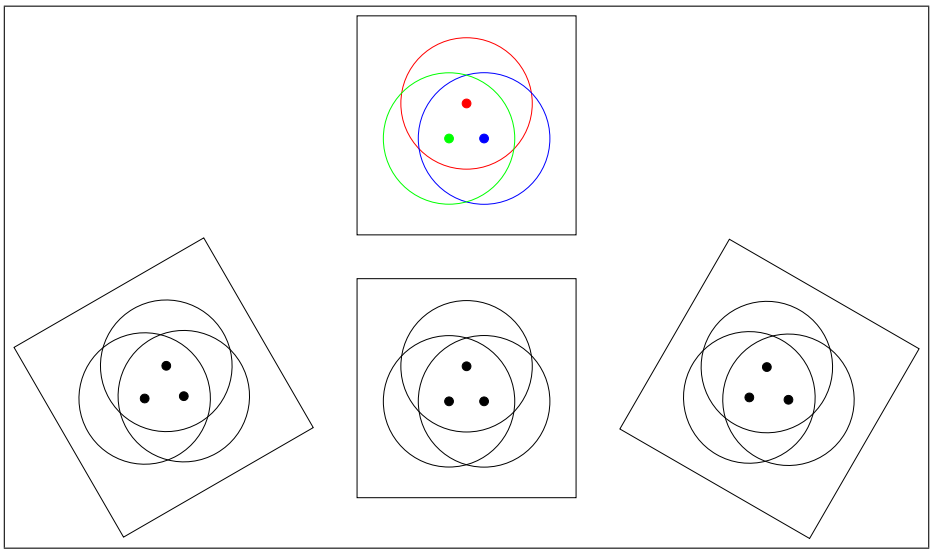


Figure 3.16: A symmetrical situation illustrating the necessity of an initial guess of the platform position for the mapping process.

3.2.4.2 Least Squares Fit

After the mapping is done the above equations may be rewritten using the real mark coordinates as:

$$\begin{aligned}
{}^{m}x_i &= {}^{p}x + d_i \cdot \sin({}^{p}\alpha + \alpha_i) \\
&= {}^{p}x + d_i \cdot (\sin {}^{p}\alpha \cos\alpha_i + \cos {}^{p}\alpha \sin\alpha_i) \\
&= {}^{p}x + \sin {}^{p}\alpha \cdot (d_i \cos\alpha_i) + \cos {}^{p}\alpha \cdot (d_i \sin\alpha_i) \\
{}^{m}y_i &= {}^{p}y + d_i \cdot \cos({}^{p}\alpha + \alpha_i) \\
&= {}^{p}y + d_i \cdot (\cos {}^{p}\alpha \cos\alpha_i - \sin {}^{p}\alpha \sin\alpha_i) \\
&= {}^{p}y + \cos {}^{p}\alpha \cdot (d_i \cos\alpha_i) - \sin {}^{p}\alpha \cdot (d_i \sin\alpha_i)
\end{aligned}$$

In order to solve this for the real platform position $({}^{p}x, {}^{p}y, {}^{p}\alpha)$ the equations can be rewritten as a *linear equation system* (LES):

$$\begin{pmatrix} {}^{m}x_i \\ {}^{m}y_i \end{pmatrix} = \begin{bmatrix} 1 & 0 & d_i \cos\alpha_i & d_i \sin\alpha_i \\ 0 & 1 & -d_i \sin\alpha_i & d_i \cos\alpha_i \end{bmatrix} \cdot \begin{pmatrix} {}^{p}x \\ {}^{p}y \\ \sin {}^{p}\alpha \\ \cos {}^{p}\alpha \end{pmatrix}$$

with $({}^{p}x, {}^{p}y, \sin {}^{p}\alpha, \cos {}^{p}\alpha)$ being the vector of unknowns. So far this LES has only two equations, but since each mapped mark contributes two equations a larger system

$$\vec{y} \quad = \quad M \cdot \vec{x}$$

can be aggregated. In general the following three cases are possible:

1. Only one mark can be seen/mapped: The matrix M is non-square and non-invertible. The LES is underdetermined and would yield an infinite number of solutions.
2. Exactly two marks can be seen/mapped: The matrix M is square and may or may not be invertible, depending on its rank[15]. If it is invertible the LES yields exactly one solution.
3. Three or more marks can be seen/mapped: The matrix M is non-square and non-invertible. The LES is overdetermined and may yield no unique solution.

These cases can best be explained at the simplified example of intersecting two-dimensional lines in a two-dimensional space as in figure 3.17. If there is only one line there is no intersection. If there are two lines there may be an intersection if the lines are not parallel. If there are three (or more) lines there may still be a single intersection, but the likelihood is that they will not intersect in one *point*.

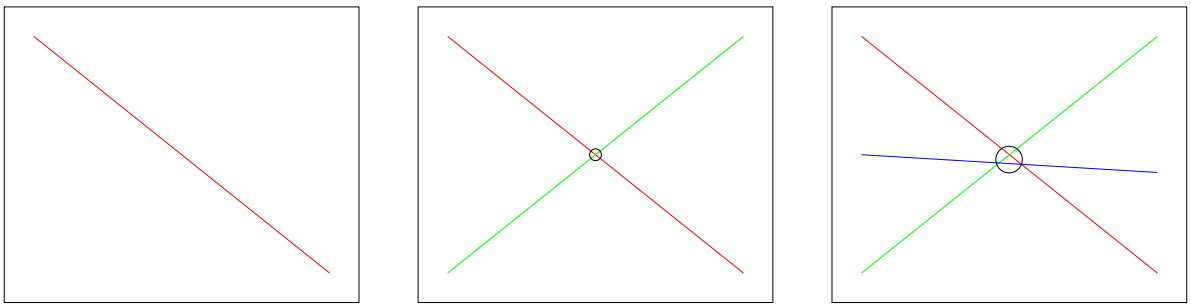


Figure 3.17: The problem of intersecting multiple lines in one point.

The problem with case three is that according to the physical model behind the problem it may be expected that these "lines" should *theoretically* intersect in one point, but since the values are derived from measurements with noise they *practically* will not. What will happen is that the intersection points of every two lines will be close together, but not identical. If no more information about the problem is given the best way is to select that solution that minimises the total squared error over all equations. This solution is called the *least squares fit* (LSF).

Instead of computing this solution iteratively a direct solution is of course preferable. One possibility is to use an approximation $M^{+} \approx M^{-1}$ like

$$M^{+} \quad = \quad (M^{T}M)^{-1}M^{T} \tag{3.10}$$

or

$$M^{+} \quad = \quad M^{T}(MM^{T})^{-1} \tag{3.11}$$

Since the goal of this M^{+} is to fulfil the condition

$$MM^{+} \quad \approx \quad I$$

[15] Since it is not intended to invert the matrix as such no further attention has been paid to this case.

as good as possible it is called the *pseudo inverse* (PI) or *Moore-Penrose Inverse* (MPI) of a matrix M.[16] This PI exists for matrices of all sizes: Equation 3.10 is for matrices with more rows than columns, and equation 3.11 is for matrices with more columns than rows. The reason for this case-split is that the "inner core" of the PI (the part of the equation that is in brackets) has to be kept small because otherwise the PI would have more free parameters than the underlying LES, which would lead to problems. For our case of more equations than unknowns equation 3.10 has to be used.

An interesting situation occurs when the PI is applied to a square matrix M. In this case the PI is actually equivalent to the normal inverse M^{-1} because for square matrices A and B the condition $(AB)^{-1} = B^{-1}A^{-1}$ holds and equation 3.10 can therefore be rewritten as:

$$\begin{aligned} M^{+} &= (M^T M)^{-1} M^T \\ M^{+} &= M^{-1} M^{T^{-1}} M^T \\ M^{+} &= M^{-1} \end{aligned}$$

Note that the PI only makes it possible to use inversion by ensuring that the matrix that is to be inverted is square, but not that the inversion itself will succeed. Although for our problem this is not the case, it is in general still possible that the argument of the inversion is a singular matrix. For problems for which this can not be avoided the *damped pseudo inverse* (DPI)

$$M^{-1} \approx (M^T M + \lambda I)^{-1} M^T$$

can be used to guarantee invertibility.

3.2.4.3 Summary

Applied to the localisation of a mobile platform the PI can be used to address the problem of solving the over-determined LES that is built by observing the landmarks. In the presented form it is, however, of only limited use because it does not take the other sensors into account. It has been used in early stages of the project to verify the original GENCONTROL software, but has soon been dropped in favour of the Kalman filter that integrates all sensors and allows predictions. Therefore no tests of the performance and accuracy of PI will be given here.

The really important issues introduced with the PI/LSF are the ideas of an *initial guess* and *tracking* of a position. It has been shown that this approach does not actually compute a position from the measurements alone, but needs an initial guess. Consequently, in the next cycle the output of the last cycle will be used as an initial guess. This remains potentially troublesome because if the association of the landmarks is ever lost it will never come back automatically and the localisation will fail, but this will be discussed in more detail in the next section.

3.2.5 Kalman Filter

The so far presented approaches are all stand-alone computations for one type of sensor, ignoring the fact that the platform has multiple sensors. These sensors provide information about different parts of the platform's *state* (position, orientation, velocity, ...), for example the odometry can be directly used to obtain a velocity and indirectly – via DR – to obtain a position, whereas the laser scanners only deliver a position. The question of how to *fuse* either these

[16] See [Moore 1920] and [Penrose 1955].

measurements or their results has so far not yet been raised. The naive approach is of course to use all approaches separately and compute and use their average output as “the position”, but that would raise a lot of questions:

- What if the quality of the different measurements differ? As has already been shown GPS lies in the range of several metres, whereas the odometry lies in the range of fractions of a millimetre. How shall these differences be incorporated?

- What if the quality of the different measurements does not stay the same over time? What if it does not stay the same over the position at which they are taken?

- What if the measurements are taken with a different sample interval? Does one have to wait until measurements from all sensors have arrived and then process them in a single batch, or can they be processed as they arrive? This may not only be a question of available computing power.

- What if it is not sufficient to know the current system state? For example, in order to map the laser marks a prediction of the current state before its measurement is perhaps not strictly necessary, but at least desirable.

These questions can only be addressed if more knowledge about the setup is taken into account. More knowledge generally means to add more parameters to the system state, for example a velocity in addition to a position to allow predictions. As a result, the complexity of the setup increases and a formal framework is needed to maintain manageability and mathematical soundness. This framework needs to be simple enough to be analysable and yet powerful enough to allow modelling of complex systems.

In 1960, Rudolph E. Kalman introduced an approach referred to as the *Kalman filter* (KF), which does exactly that [Kalman 1960]. The KF in general is a mathematical method for linear filtering of discrete data. It is a recursive algorithm to *estimate* the state of a system according to a set of measurements, given a linear model of how they relate. The emphasis is on “estimate” because the “correct” state is unknown because both the state and the measurements are allowed to be superimposed by gaussian white noise, and because the linear model may contain approximation errors. The KF does not see the superimposed noise as a disadvantage, but in fact tries to estimate it too and use it for weighing the different measurements. The resulting statistically optimal state estimation is therefore precisely that state that meets the least-squares criterion.

In this section only a brief introduction into Kalman Filters will be given. For a more detailed workout see [Schneider and Westhoff 2002], from which parts of this introduction have been taken.

3.2.5.1 System and Measurement Model

The KF as a black box is based on the assumption of a linear world model. Furthermore, the world is assumed to be a time-discrete one. In such a world the *system* is modelled as a *state vector*

$$\vec{x}_t = \begin{pmatrix} x_1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}_t$$

representing the individual components of the system state and a *system equation*

$$\boxed{\vec{x}_{t+1} = F\vec{x}_t + B\vec{u}_t + \vec{w}_t} \tag{3.12}$$

realising the projection of the system state from time step t to time step $t+1$. The matrix F is called the *system matrix* and $\vec{w}_t$ the noise that affects the system in each time step. The vector $\vec{u}_t$ is the control input of the system used to influence the system via the matrix B. In the following the term $B\vec{u}_t$ is dropped for the sake of simplicity without loss of generality.

The *measurement* is modelled as a *measurement vector*

$$\vec{z}_t = \begin{pmatrix} z_1 \\ z_1 \\ \vdots \\ z_m \end{pmatrix}_t$$

representing the individual measurements and a *measurement equation*

$$\boxed{\vec{z}_t = H\vec{x}_t + \vec{v}_t} \tag{3.13}$$

linking the measurement to the system state. The matrix H is called the *measurement matrix* and $\vec{v}_t$ is the noise that affects the measurement. A block diagram of the KF can be seen in figure 3.18.

The noise terms $\vec{w}_t$ and $\vec{v}_t$ only represent the white, uncorrelated statistical part of the noise of the system and the measurement, not any systematic effects. They can therefore be described by Gaussian distributions with zero-mean $\mathcal{N}(0, Q)$, where $Q = E\{\vec{w}_t\vec{w}_t^T\} = \text{diag}(q_1, \ldots, q_n)$ and $\mathcal{N}(0, R)$, where $R = E\{\vec{v}_t\vec{v}_t^T\} = \text{diag}(r_1, \ldots, r_m)$. Q and R are positive (semi-) definite matrices. The system and measurement noise must not be correlated, i.e. $E\{\vec{w}_t\vec{v}_t^T\} = 0$ holds.

In addition to $\vec{w}_t$ and $\vec{v}_t$ there is another important component dealing with statistical effects. Since the KF is going to compute estimates of the system state, that state itself has to be considered inaccurate or noisy. Since it can not be assumed that the noise in the state components is not correlated an additional matrix P is therefore used as the *covariance matrix* of the state.

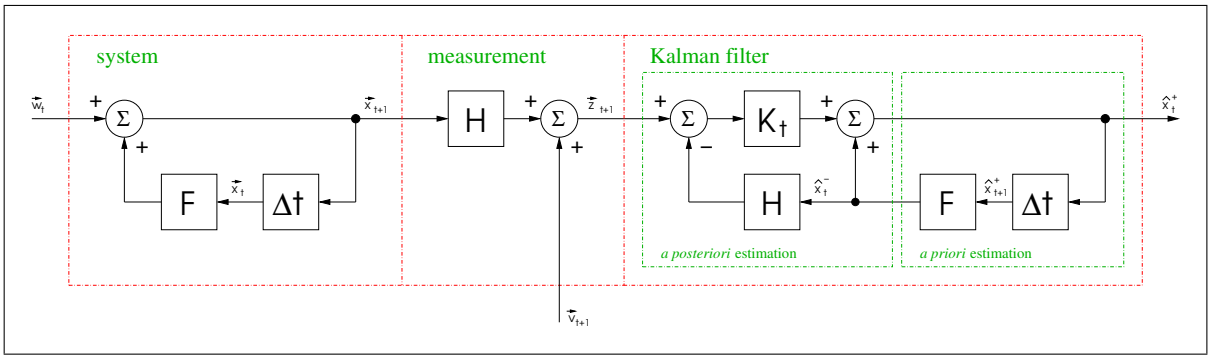


Figure 3.18: Block diagram of a Kalman filter.

3.2.5.2 Predict and Update

The sequence of operation for a KF consists of two major steps:

1. The first step is the *predict* step in which an *a-priori* estimation $\hat{x}_t^-$ of the system state just before a measurement is computed, based on the last estimate $\hat{x}_{t-1}^+$ as a substitute for the unknown real state $\vec{x}_{t-1}$.

$$\boxed{\hat{x}_t^- = F\hat{x}_{t-1}^+} \tag{3.14}$$

   After this, the covariance matrix P expressing the uncertainty has to be updated. For each predict step the variances increase (Q is added) because with no measurement to verify the situation the estimation becomes more and more unreliable.

$$\boxed{P_{t+1}^- = FP_t^+F^T + Q} \tag{3.15}$$

2. The second step is the *update* step in which a new measurement is used to verify the prediction. First a matrix K_t called the *Kalman matrix* is computed. It uses the uncertainty (covariances) of state components and the measurement model to estimate the uncertainty of the measurement components and use them as gain factors for the update. This way those sensors which are considered most accurate are given the highest weights.

$$\boxed{K_t = P_t^- H^T \left(HP_t^- H^T + R\right)^{-1}} \tag{3.16}$$

Then an *a-posteriori* estimation $\hat{x}_t^+$ of the system state is computed based on that matrix and a measurement $\vec{z}_t$

$$\boxed{\hat{x}_t^+ = \hat{x}_t^- + K_t(\underbrace{\vec{z}_t - H\hat{x}_t^-}_{\vec{i}_t})} \tag{3.17}$$

The vector $\vec{i}_t$ is called the *innovation* – the difference between the measurement and the prediction of the measurement according to the prediction of the state. If the state prediction was correct the innovation will be 0 and the state prediction will not be changed.

Finally, an a-posteriori estimate of the covariance matrix is computed as the starting point of a new cycle.

$$\boxed{P_t^+ = (I - K_t H)\, P_t^-} \tag{3.18}$$

See [Schneider and Westhoff 2002] or [Bar-Shalom and Li 1993] for a derivation of these equations.

This sequence of equations 3.14 to 3.17 is called iteratively for each cycle of the time-discrete system. If different measurements should be available at different intervals, the cycle can be split and a separate measurement model and update cycle can be used for each measurement.

Also, none of the matrices F, H, Q and R have to be constant over time. For example, in case of a moving object the state will have to hold its velocity. In order to compute predictions of the state that velocity has to be multiplied by the elapsed time Δt, and this can only be done in a KF by writing this Δt in the system model F. Since the Δt may vary F is therefore no longer constant.

Having variable Q and R means that the accuracy of the system or the measurement changes over time. Often only a poor initial guess about these accuracies exists and the reliability of the system can be improved when they are changed at run-time. For example different laser marks may be considered unequally reliable depending on how long they have been observed. A mark that only flashed up once can well be considered unsafe compared to a mark that has been constantly observed over a long time. Appropriate individual weights are therefore a worthy consideration.

All these manipulations are not part of the KF and have to go into modifying the matrices.

3.2.5.3 Summary

The KF is a construct to estimate and track the state of a linear time-discrete differential system. If a linear approximation of a system and the measurements are feasible and the variances of the sensors are at least approximately known the KF can be proven to be the statistically optimal and stable estimator of the system because it implements an iterative least-squares fit. It can then compute past, current and future states of the system.

On the other hand side, the KF in general as well as in particular for a mobile platform does raise a few issues, which will be shown in detail in section 3.2.8. Summed up in short words the general KF may fail and produce appearingly random output if the preconditions are not met. In addition to that, using a KF for a mobile platform where the state model contains an integrator invokes problems because integrators are not stable in the sense of "bounded input" $\rightarrow$ "bounded output"[17].

3.2.6 Extended Kalman Filter

The KF uses *linear* models for the system and the measurement and assumes *gaussian* distributions of the errors. Often a situation is more complex than that and can only described by more general functions. In the most generic case the state $\vec{x}_t$ and the measurement $\vec{z}_t$ in such a situation remain as with the plain KF, but the models allow arbitrary functions $f(\vec{x})$ and $h(\vec{x})$ instead of matrices F and H. That is, the system equation 3.12 changes from

$$\vec{x}_{t+1} = F\vec{x}_t + B\vec{u}_t + \vec{w}_t$$

into

$$\vec{x}_{t+1} = f(\vec{x}_t, \vec{u}_t, \vec{w}_t) \tag{3.19}$$

and the measurement equation 3.13 changes from

$$\vec{z}_t = H\vec{x}_t + \vec{v}_t$$

into

$$\vec{z}_t = h(\vec{x}_t, \vec{v}_t) \tag{3.20}$$

where $\vec{u}_t$ is the controller input. The error distributions for these models must not necessarily be gaussian, but can be of any shape.

3.2.6.1 Simplifications

Solving such a generic problem is not always possible, so simplifications have been sought. The *extended Kalman filter* (EKF) is one such extension [Schmidt 1970] [Schmidt 1976]. The EKF handles the non-linear models f and h by approximating them by *taylor series* expansions, and the error distributions $\vec{w}_t$ and $\vec{v}_t$ by approximating them by gaussian distributions with covariance matrices Q and R. Depending on the number of terms of the taylor series expansion after which one abandons the approximation there exist several different EKFs. The simplest one described here is the *first order* EKF, which only uses a linear approximation of the models. The models f and h are therefore approximated by their *Jacobian* matrices F^J and H^J.

The sequence of operations for the EKF is shown in figure 3.19. It corresponds with the sequence of operations for the plain KF:

1. Prediction: First the new predicted estimate $\hat{x}^-_{t+1}$ of the state is computed as

$$\boxed{\hat{x}^-_{t+1} = f(\hat{x}^+_t, \vec{u})} \tag{3.21}$$

[17] See [Bar-Shalom and Li 1993].

and then the covariance matrix of the a-priori estimation error as

$$\boxed{P_{t+1}^- = F_t^J P_t^+ (F_t^J)^T + W_t^J Q (W_t^J)^T} \tag{3.22}$$

2. Update with measurements: First the Kalman matrix K_t is computed as

$$\boxed{K_t = P_t^- (H_t^J)^T \left(H_t^J P_t^- (H_t^J)^T + V_t^J R (V_t^J)^T \right)^{-1}} \tag{3.23}$$

then the a-posteriori estimate $\hat{x}_t^+$ of the state as

$$\boxed{\hat{x}_t^+ = \hat{x}_t^- + K_t(\vec{z}_t - h(\hat{x}_t^-))} \tag{3.24}$$

and finally the covariance matrix P_t^+ of the a-posteriori estimation

$$\boxed{P_t^+ = (I - K_t H_t^J) P_t^-} \tag{3.25}$$

The errors $\vec{w}_t$ and $\vec{v}_t$ have for simplicity been assumed to be 0 in these equations – a legal assumption as they are only required to have a zero mean and no more knowledge about them is known at this stage.

The resulting EKF is more complex than a plain KF and therefore the question of its stability has to be raised. Bar-Shalom and Li say[18] that

> *In general, a nonlinear transformation will introduce a bias and the covariance calculation based on a series expansion is not always accurate. (...) In practice, if the initial error and the noises are not too large, then the EKF performs well. The actual limits of the successful use of the linearisation techniques implicit in the EKF can be obtained only via extensive Monte Carlo simulations for consistency verification.*

and

> *The sufficient conditions for the stability of the KF are not necessarily sufficient for the EKF. The reason is that its inherent approximations can lead to divergence of the filter – unbounded estimation errors.*

[18] [Bar-Shalom and Li 1993], chapter 10.3.3, page 388

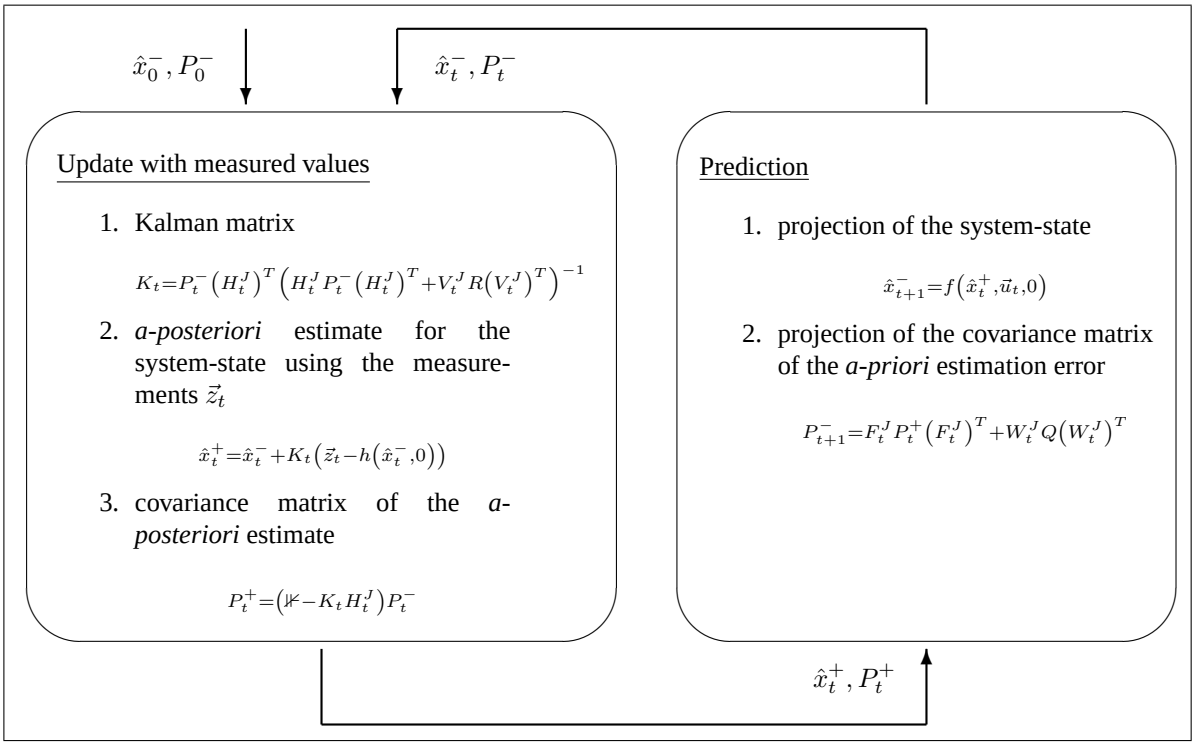


Figure 3.19: Sequence of operations for the extended Kalman filter.

Once modelling errors are assumed to be the cause of problems using a more accurate model is of course the next idea, and in case of a taylor series expansion this means to use more terms. A *second order* EKF would not only use the function's Jacobian, but also the *Hessian*. Although this appears to be more accurate then a first order EKF, the result is *"generally reported be only partly successful"* [19]. Bar-Shalom and Li express the same problem in clearer words[20]: *"There is no guarantee that even the second order terms can compensate for (...) errors"*. Second or higher order filters have therefore not been researched here.

3.2.6.2 Summary

The EKF is an enhancement to the KF which allows non-linear models like the ones needed for our mobile platform. It uses one (or more) terms from the taylor series expansion to simplify these models, leading to a linear (in case of just one term) approximator of the non-linear models. It is, however, this approximation which makes it impossible to prove the filter's stability. Extensive tests about the filter's behaviour are therefore necessary. Apart from this, the EKF is an interesting state estimator because it is not too computationally expensive.

3.2.7 The MP-L655 Extended Kalman Filter

After generally establishing the EKF as a suitable means of estimating the state of a mobile platform the concrete filter for the MP-L655 shall now be discussed. Schneider and Westhoff[21] have developed a complete kinematic and (partially) dynamic, down to including differential

[19] [Wright, Maskell and Briers 2003]
[20] [Bar-Shalom and Li 1993], chapter 10.3.3, page 388
[21] [Schneider and Westhoff 2002], chapter 4.

equations for a model of the drive system. Their overall model is very detailed and complex, and so only a brief sketch containing their most important results will be given here.

3.2.7.1 State Definitions

Schneider and Westhoff have chosen a model which not only consists of the platform's position and velocity, but also includes the platform's accelerations, the angular wheel velocities and the positions of the laser marks. The complete state vector $\vec{x}$ is defined as:

$$\vec{x} = \left(\omega_R, \omega_L, \dot{x}, \dot{y}, \dot{\phi}, \ddot{x}, \ddot{y}, \ddot{\phi}, x, y, \phi, x_{f_1}, y_{f_1}, ..., x_{f_n}, y_{f_n}\right)^T$$

with ω_R and ω_L being the wheel velocities as modelled according to a first order differential equation, $(\dot{x}, \dot{y}, \dot{\phi})$ the current platform velocity as derived from the wheel velocities, $(\ddot{x}, \ddot{y}, \ddot{\phi})$ the current platform acceleration as derived from differences in the velocity, (x, y, ϕ) the current platform position as derived from integrating the velocity and acceleration and (x_{f_i}, y_{f_i}) the coordinates of the laser mark (feature) i.

Note that the velocity of the platform is not modelled as a plain translational and rotational velocity, but as separate x and y-components of the the translation. This means that the equations for them always have to take the current orientation ϕ into account and that they keep changing if the platform is moving a circle with constant real "velocity". Although it may appear more intuitive to regard the platform velocity as a function of the wheel velocities as constant in case of the circle, it makes computing the positions slightly more difficult. Overall there is no systematic difference between both approaches.

Having the positions (x_{f_i}, y_{f_i}) of the laser marks in the state allows them to be changed during an update, which is both an interesting and a potentially dangerous idea. Interesting, because it allows the EKF to make corrections if the position of one mark should be slightly wrong in the map. Dangerous, because these corrections are not at all bound – in the worst case all marks can be moved by an amount that no mapping to the real marks will be possible any more. Since this only adds to the fact that the EKF is *per-se* not proven stable it is not considered an urgent problem here. It remains an open question, though.

As for the measurements, there are two distinct measurement vectors for two distinct update routines. This is because the odometry and the laser scanner deliver their data at different rates and therefore have been separated. The odometry measurement is defined as

$$\vec{z}_{\text{odo}} = \left(\omega_{\text{odo}_R}, \omega_{\text{odo}_L}, \dot{\phi}_{\text{gyro}}\right)^T$$

with ω_{odo_R} and ω_{odo_L} being the wheel velocities as reported by the encoders and $\dot{\phi}_{\text{gyro}}$ the rotational velocity as reported by the gyro.

The laser measurement is defined as

$$\vec{z}_{\text{laser}} = (d_{f_1}, \alpha_{f_1}, ..., d_{f_m}, \alpha_{f_m})^T$$

with d_{f_i} being the distance to laser mark i and α_{f_i} the angle under which it is seen (in platform coordinates).

3.2.7.2 Models

The system and measurement models have to take all these state components into account, yet they are very straightforward. The system model $\hat{x}_{t+1} = f(\hat{x}_t, \vec{u})$ is given as

$$\left.\begin{array}{rcl} \omega_{t+1_R} & = & u_R + (\omega_{t_R} - u_R) \cdot e^{-\frac{\Delta t}{\tau}} \\ \omega_{t+1_L} & = & u_L + (\omega_{t_L} - u_L) \cdot e^{-\frac{\Delta t}{\tau}} \\ \dot{x}_{t+1} & = & \frac{\omega_{t+1_R} + \omega_{t+1_L}}{2} \cdot r \cdot \cos\phi_t \\ \dot{y}_{t+1} & = & \frac{\omega_{t+1_R} + \omega_{t+1_L}}{2} \cdot r \cdot \sin\phi_t \\ \dot{\phi}_{t+1} & = & \frac{\omega_{t+1_R} - \omega_{t+1_L}}{2} \cdot \frac{r}{a} \\ x_{t+1} & = & x_t + \dot{x}_{t+1} \cdot \Delta t + \frac{\ddot{x}_t}{2} \cdot \Delta t^2 \\ y_{t+1} & = & y_t + \dot{y}_{t+1} \cdot \Delta t + \frac{\ddot{y}_t}{2} \cdot \Delta t^2 \\ \phi_{t+1} & = & \phi_t + \dot{\phi}_{t+1} \cdot \Delta t + \frac{\ddot{\phi}_t}{2} \cdot \Delta t^2 \\ \ddot{x}_{t+1} & = & \frac{\dot{x}_{t+1} - \dot{x}_t}{\Delta t} \\ \ddot{y}_{t+1} & = & \frac{\dot{y}_{t+1} - \dot{y}_t}{\Delta t} \\ \ddot{\phi}_{t+1} & = & \frac{\dot{\phi}_{t+1} - \dot{\phi}_t}{\Delta t} \end{array}\right\} \tag{3.26}$$

with $\vec{u}$ being the controller input from the last cycle (the value it was commanded to reach), Δt the time interval that has elapsed since the last update, τ the time constant for the drive system model, r the drive wheel radius' and a the distance of the drive wheel from the platform centre.

The measurement models are even more intuitive. The odometry model $\hat{z} = h(\hat{x}_t)$ is given as

$$\left.\begin{array}{rcl} \omega_R & = & \omega_{t_R} \\ \omega_L & = & \omega_{t_L} \\ \dot{\phi} & = & \dot{\phi}_t \end{array}\right\} \tag{3.27}$$

and the laser model $\hat{z} = h(\hat{x}_t)$ is given as

$$\left.\begin{array}{rcl} d_i & = & \sqrt{(y_{f_i} - y_t)^2 + (x_{f_i} - x_t)^2} \\ \alpha_i & = & \operatorname{atan2}\left(y_{f_i} - y_t, x_{f_i} - x_t\right) \end{array}\right\} \tag{3.28}$$

Note that the α_i in this form is still in world coordinates and needs a proper conversion into platform coordinates.

The Jacobians shall also be briefly introduced. The system jacobian F_t^{J} is given as

$$
F_t^{\mathrm{J}} = \left[\begin{array}{cccccccccccc|cc}
e^{c_t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & e^{c_t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}e^{c_t}\cos\phi_t & \frac{r}{2}e^{c_t}\cos\phi_t & 0 & 0 & c_y\Delta t & 0 & 0 & c_y\frac{\Delta t^2}{2} & 0 & 0 & c_y & 0 & 0 \\
\frac{r}{2}e^{c_t}\sin\phi_t & \frac{r}{2}e^{c_t}\sin\phi_t & 0 & 0 & c_x\Delta t & 0 & 0 & c_x\frac{\Delta t^2}{2} & 0 & 0 & c_x & 0 & 0 \\
\frac{r}{2l_a}e^{c_t} & -\frac{r}{2l_a}e^{c_t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}e^{c_t}\cos\phi_t & \frac{r}{2\Delta t}e^{c_t}\cos\phi_t & \frac{-1}{\Delta t} & 0 & c_y & 0 & 0 & c_y\frac{\Delta t}{2} & 0 & 0 & \frac{c_y}{\Delta t} & 0 & 0 \\
\frac{r}{2\Delta t}e^{c_t}\sin\phi_t & \frac{r}{2\Delta t}e^{c_t}\sin\phi_t & 0 & \frac{-1}{\Delta t} & c_x & 0 & 0 & c_x\frac{\Delta t}{2} & 0 & 0 & \frac{c_x}{\Delta t} & 0 & 0 \\
\frac{r}{2l_a\Delta t}e^{c_t} & -\frac{r}{2l_a\Delta t}e^{c_t} & 0 & 0 & \frac{-1}{\Delta t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]
$$

where

$$
\begin{aligned}
c_t &= \frac{-\Delta t}{\tau} \\
c_x &= \dot{x}_{t+1} - \mathrm{w}_3 \\
c_y &= -\dot{y}_{t+1} + \mathrm{w}_4
\end{aligned}
$$

Note that this F_t^{J} only contains two rows and columns for one laser mark, indicated by lines separating them. The actual F_t^{J} is of course larger because it has to hold all laser marks, but the additional rows and columns are only filled according to the one given here and have therefore been left out for simplicity.

The measurement jacobian H_t^{J} is given as

$$
H_t^{\mathrm{J}} = \left[\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right.
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{x_t - x_{f_i,t}}{d} & \frac{y_t - y_{f_i,t}}{d} & 0 & \frac{x_{f_i,t} - x_t}{d} \quad \frac{y_{f_i,t} - y_t}{d} \\
\frac{y_{f_i,t} - y_t}{d} & \frac{x_t - x_{f_i,t}}{d} & -1 & \frac{y_t - y_{f_i,t}}{d} \quad \frac{x_{f_i,t} - x_t}{d}
\end{array}
\left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right]
$$

$$
d = \sqrt{(x_t - x_{f_i,t})^2 + (y_t - y_{f_i,t})^2}
$$

The system noise jacobian W_t^{J} is given as

$$W_t^{\mathrm{J}} = \left[\begin{array}{cc|cccccccccc|cc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}\cos\phi_t & \frac{r}{2}\cos\phi_t & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}\sin\phi_t & \frac{r}{2}\sin\phi_t & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2l_a} & \frac{-r}{2l_a} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}\cos\phi_t & \frac{r}{2\Delta t}\cos\phi_t & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}\sin\phi_t & \frac{r}{2\Delta t}\sin\phi_t & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2l_a\Delta t} & \frac{-r}{2l_a\Delta t} & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

And, finally, the measurement noise jacobian V_t^{J} is given as

$$V_t^{\mathrm{J}} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

With these equations the EKF as in equations 3.21 to 3.25 is fully specified. However, this specification is not what was finally implemented because a minor change has been found necessary.

3.2.7.3 Model without Acceleration

Although theoretically correct the complete model has been discovered to be troublesome during testing. It was found that the acceleration introducing terms that contain $\frac{1}{\Delta t}$ or $\frac{1}{\Delta t^2}$ caused problems when Δt got too small. Two main reasons for this problem could be identified:

First, since the underlying Linux kernel is not really real-time capable[22] it may well happen that the cycle timing is not strictly fixed. Whereas usually the deviations from the desired timing are quite tolerable they are – by design of the linux kernel – unbound under worst circumstances. In addition to the general problems of measuring velocities and accelerations based on varying time intervals it also means that Δt can become arbitrarily small.

Second, since the sensors deliver their data with different rates the odometry and laser update are implemented as different routines to be called at different times. This raises the idea to also use a different predict for each of the updates. The EKF does work when there is only one predict step at a time, but yields only suboptimal results in this case. On the other hand side,

[22]The system still uses a Linux kernel from the 2.4 series, although the 2.6 series is supposed to be better in this aspect.

with an individual predict for each update the filter can become numerically unstable if the time Δt between these predicts becomes arbitrarily small. This situation is visualised in figure 3.20.

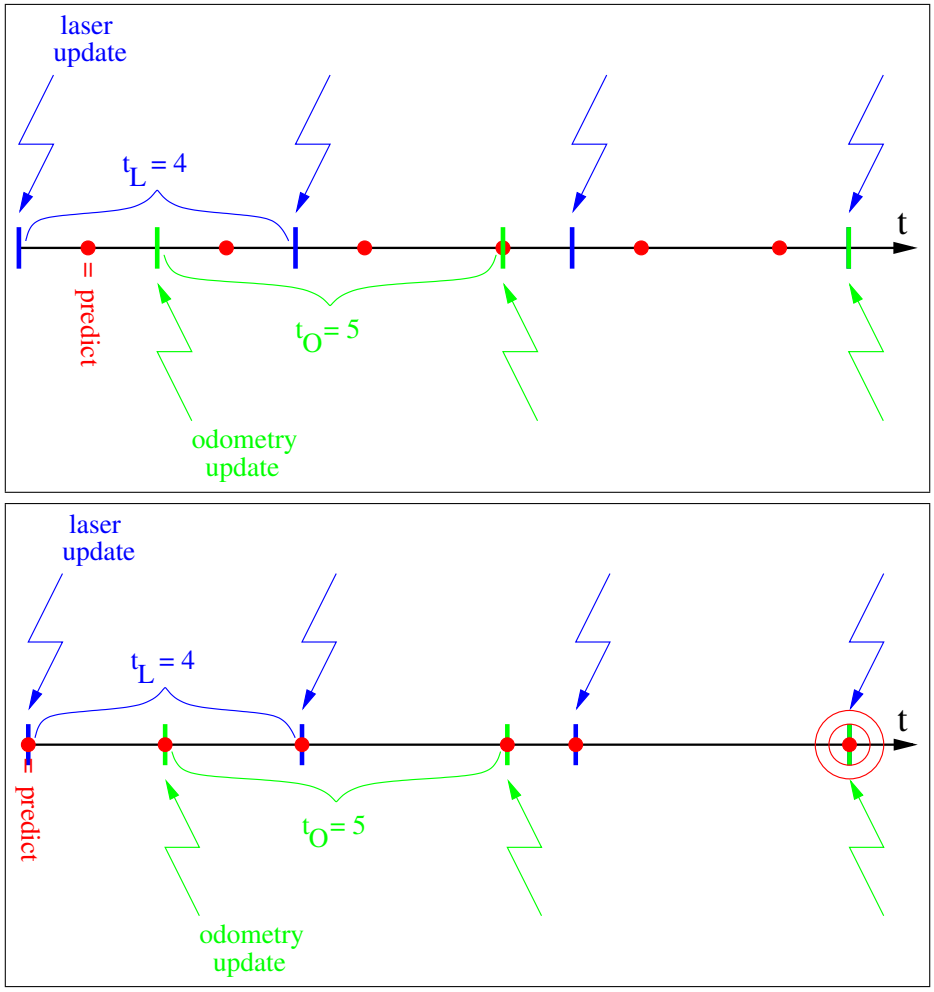


Figure 3.20: A predict step asynchronous to the updates (upper image) means that the update steps will never perfectly fit to the predicted system state and the filter will yield suboptimal results. A predict step called synchronous with each update (lower image) means that the time Δt between two predicts can become arbitrarily small, which can cause numerical problems with the system model.

In both cases components of the covariance matrix P can become large because of divisions by Δt or Δt^2, which leads to that components of the Kalman matrix K become large which in turns leads to that large "corrections" are done in the update step. Practically this means that the position after an update may be wrong by an amount larger than the catch radius used for mark mapping, so that in the next cycle the mapping of the landmarks is no longer possible. This is consistent with the fact that the EKF can become unstable if the amount of noise on the data is too high.

As a result of this the parts of the system model f from equation 3.26 that contain acceleration terms have been shortened to

$$
\begin{aligned}
x_{t+1} &= x_t + \dot{x}_{t+1} \cdot \Delta t \\
y_{t+1} &= y_t + \dot{y}_{t+1} \cdot \Delta t \\
\phi_{t+1} &= \phi_t + \dot{\phi}_{t+1} \cdot \Delta t \\
\ddot{x}_{t+1} &= 0 \\
\ddot{y}_{t+1} &= 0 \\
\ddot{\phi}_{t+1} &= 0
\end{aligned}
$$

virtually disabling accelerations for the other parts. Also, the system jacobian F_t^{J} has been cleaned of the appropriate entries.

3.2.8 Kalman Filtering Summary

Although often used for estimation problems the (E)KF does raise a few issues, which will be discussed in this subsection. The issues are partially inherent to the KF and not specific to this work. They are grouped in general issues, issues concerning the localisation and numerical issues. If ignored, they can make the filter unstable.

3.2.8.1 General Issues

First of all, it has to be emphasised that the KF only works for (linear) discrete time dynamic system. If the system to be handled is not a dynamic system the KF can not be applied. Practically this means that the state has to hold an entity (the position) as well as the first derivative of that entity (the velocity). This may sound too obvious, but there is a possibility to see a mobile platform as a static object: The odometry as well as the laser mark recognition can be implemented in a way that they yield absolute positions without ever using a platform velocity. A KF can not work without a velocity because without a velocity no predictions are possible, and without these predictions no estimates of the measurements are possible, and without these estimates no weighted update is possible.

Another aspect is that the KF is only an *estimator* in several ways, i.e. it will never tell the "truth". The primary reason for this is that the truth is unknown because no *precise* measurement is given – all measurements are all superimposed by noise. If there was a precise and reliable measurement there would be no need to use a KF in the first place. Expecting a KF to give absolutely "correct" values is simply out of its scope.

The secondary reason is that the KF uses gaussian shaped probability distributions to fuse the predicted system state and the measurements into the most likeliest new system state. This is only an approximation because – depending on the system – these entities may in reality have arbitrarily shaped probability distributions. The KF assumes that it is legal to use this approximation in order to simplify computations, but it also means that it can no longer yield the "correct" values. Approaches like the UKF or particle filter do exist to overcome this drawback by not assuming gaussian shaped distributions, but take much more computation time (see section 3.5.2).

The tertiary reason is that the KF only uses simplified *models* of the system and measurements. It further assumes the measurements to basically comply to these models and only be disturbed random white noise. This assumption does have an impact so important that it needs special attention.

As long as the system and the measurement comply to the models the KF is known to be a *statistically optimal estimator*, which means that it computes the "optimal" state by weighing the prediction and the measurements according to their variances. The resulting output is still affected by noise, but since the KF gains certainty from fusing multiple measurements the noise of the output is supposed to be smaller than the noise of any of the individual measurements. An unwary user may therefore in particular expect the filter output to not jump by arbitrary amounts. This, however, only holds as long as individual measurements comply to the model and do not contradict each other.

If two measurements yield contradicting results (e.g. because one sensor is broken) the KF can do nothing but apply its models – it does not *see* the contradiction and therefore can not do anything against it. As a result it will produce a suboptimal output that may even jump by an unexpected amount. This especially applies to mobile robots because their reality is very complex and their models usually very simple. In most cases they are designed to cover the "normal" conditions only, but not any errors.

For example, if there has been a bump in the floor and the gyro has seen a jerk sideways where the odometry thinks the platform is still going straight ahead, two measurements contradict each other. The KF is not able to determine which measurement is wrong, so all it can do is apply its models. The resulting output will clearly be "wrong" because it is only a superimposition of the two contradicting measurements[23]. There is no such thing as a covariance matrix of the measurement which the KF itself can change to compensate for the contradiction because there is no entity which determines which input was wrong.

If such a situation only affects one measurement and occurs only once it may not have a lasting effect. If it does, however, affect more measurements or occurs more frequently it has to be handled either in the model (which makes the model bigger and in turns potentially more troublesome) or by modifying the measurement noise R (which requires external pre- or post-processing).

In general a KF will return to the "correct" solution as soon as the measurements are in accordance again, but in case of a mobile platform this may not necessarily be the case. On the MP-L655 an "adequate" state prediction is needed to map the laser marks as part of the measurement, and this mapping can fail in the above situation if the error in the state is too large. As a result no more laser measurements will be possible from this point on into the future and the KF – now only relying on odometry – will become arbitrarily wrong. This is the absolute worst case that can happen and all the statistical optimality does not help here because it is a problem of the model, not the KF. The only way to deal with such a situation is to have other (lower priority) absolute position computation(s) running in parallel, verifying and resetting the KF state if necessary.

[23]This is even more complicated by the fact that for the MP-L655 the system model does not allow sideway motions, but the measurement model does. The reason to exclude them from the system model is that according to its kinematic the MP-L655 can not produce these motions with its own motors, and the reason to include them in the measurement model is that there must be a possibility to correct them if they erroneously do occur. If, for example, one drive motor spins freely because it gets stuck in a drain in the floor the kinematic suddenly changes. Contrary to the usual kinematic which can only produce rotations about points on the axis through the two drive wheels the new kinematic using the remaining drive wheel and the castor wheels also allows rotations about other points. The resulting motion therefore can have a non-zero sideways component, and since the system model does not incorporate it the measurement model must have a chance to adjust the state accordingly.

A final aspect is that in case of the plain KF the models also have to be *linear*, for example the measurement has to be a *linear* function of the state. Whereas in a lot of cases a linear approximation of a non-linear system is a feasible first-order approach, there are systems for which this does not work.

Unfortunately, one example of such a system is one that involves multidimensional distances and angles like a mobile platform. As for the distances, if the state contains the cartesian positions of the platform and the marks and the laser scanner measures an angle and a distance then the measurement model would have to contain terms like

$$d_{\text{mark}} = \sqrt{\vec{p}^{\,2}_{\text{platform}} - \vec{p}^{\,2}_{\text{mark}}}$$

which are clearly non-linear. As for the angles the problem is the same. If the state contains the orientation of the platform as an angle α (and not its sine and cosine) and the odometry yields a cartesian velocity v_{odo} in heading direction the measurement model would have to contain terms (in C++ style) like

$$\begin{aligned} p_x &+= v_{\text{odo}} \cdot T_{\text{Sample}} \cdot \sin\alpha \\ p_y &+= v_{\text{odo}} \cdot T_{\text{Sample}} \cdot \cos\alpha \end{aligned}$$

which are again non-linear. The question of whether a linear approximation of one or the other term is feasible will not be raised here.

Searching the mapping is also a non-linear process, so in order to map the laser marks a KF always needs additional preprocessing for a mobile platform.

The EKF in turns allows non-linear models, but is more complex and can not be proven to be stable even with bounded errors. One reason for this is that the system and measurement noise are still approximated by gaussian distributions only, which is a simplification that leads to suboptimal (and in some cases unstable) behaviour. If the distribution of any component would for example have the shape of a banana an approximation with a gaussian distribution can not express all information in that distribution (see figure 3.21).

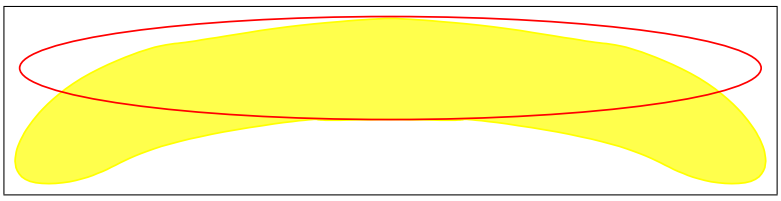


Figure 3.21: Approximating a banana with a gaussian distribution.

Such a situation actually occurs with the odometry: Consider the measurement of the drive wheel velocity to be affected by pure gaussian noise, which is – without any further knowledge – a legal assumption. If the forward kinematic (DR) is applied to this measurement to obtain a position after some small time of motion the probability distribution of this position will have exactly that banana shape.

It also occurs with the laser measurements during rotations: Due to wheel slippage when accelerating too fast it is quite frequent that the platform has not rotated as much as the prediction yielded. On the other hand it will probably not have moved too much away from the

point it was standing at. This means that the distance to the marks is likely to be the same as predicted, but the angles under which they are seen are not. The resulting distribution of the laser measurement prediction has again exactly the shape of a banana.

A (E)KF that does not take this into account will yield suboptimal estimations, if any.

3.2.8.2 Localisation Issues

According to the (E)KF the *initial state* is an unknown random variable that is supposed to converge. A mobile platform, however, needs an adequate initial state because its measurement involves mapping the laser marks. Because of this mapping the error of the state must be bound to some absolute value, even though the (E)KF stability itself might not be affected by sporadic outliers. If no marks can be mapped no unique laser update is possible, a situation which is unlikely to cure itself in the following cycles. The question of the allowable noise is therefore not a question of the filter gradually degrading, but of completely and abruptly failing to work.

In case of no unique mapping according to the current state estimate it is still possible that there are several solutions distinct from the current state which might "fit" (with different least-squares errors). In such a case a *multi-hypothesis approach* can be used to track these solutions during some exploratory movements until one of them becomes preferable according to its covariance expressing the certainty. These approaches are not suitable for the laboratory setup, precisely because the laser scanner only scans a single plane of the room and therefore the exploratory movements can not be guaranteed to not lead to collisions. Other sensors like ultrasonic sensors that monitor not only planes but true spaces are needed to guarantee this. Since these sensors do not exist on the platform these ideas have not been pursued.

The matrices Q and R expressing the covariances of the *system* and *measurement noise* of $\vec{w}_t$ and $\vec{v}_t$ are very important as they are the only parameters over which one can adjust the (E)KF. On the other hand side they are often unknown and only approximated or guessed. This means that the (E)KF uses suboptimal knowledge about the accuracy of sensors and therefore produces suboptimal state estimates. If one sensor is erroneously given a too large variance this means that its contribution to the state estimation is weighed too low compared to others. If these other sensors in turns do have a higher variance this means that the resulting state estimation will be noisier than it would have to be if the measurement variances had been correct. The worst possible case is to not include any knowledge about the sensors at all and have all entries in the matrices as constant 1.

Let, for example, the system be a human being and the state the information of where "up" and "down" are. A human has two main sensors for detecting up and down. The primary sensor is the vestibule[24] in the ear ("gravity organ") which physically detects the pull of gravity. However, this organ may be disturbed for example by a severe cold or flu. In such a case a human can still tell up from down by using its eyes in combination with knowledge of how the environment is *expected* to look like if everything is ok. Biologic evolution has in the course of history adjusted the variances in order to put more emphasis on the eyes than on the gravity organ, probably because of some evolutionary advantage. Only when both sensors are not in agreement the human becomes aware that there actually are two sensors, for example in

[24]The vestibule's *utricle* and *sacculae* are used for sensing static head positions, whereas the semicircular canals are used for sensing motions in each of the 3 spatial planes. `http://webschoolsolutions.com/patts/systems/ear.htm`

deliberately tilted houses where rooms appear normal but everyone is standing lopsided or areas where water seems to flow uphill[25].

Another example is that of a mobile platform and its sensors: Too much focus on the laser scanners means that even a single measurement error strikes through to the position estimation (and the association of the laser marks might be lost). On the other hand too few focus on them means that slippage might not be detected (and the association of the laser marks might be lost). Any mechanism to deal with this problem can only adjust the variances $\vec{w}_t$ and $\vec{v}_t$ on-line. If a symptom can not be cured by this, the system can not be approximated by an (E)KF.

3.2.8.3 Numerical Issues

The (E)KF is a construct that has undergone a lot of extensive analysis since its invention. It can be proven to be stable if some conditions on - amongst other things - the covariance matrix are given[26]. This raises a few issues when implementing an (E)KF on a computer with limited numerical resolution. Though never having been observed at our setup these issues can in the worst case make the (E)KF unstable and shall therefore at least be mentioned here.

The first issue is that of numerical *stability* of the computation itself. Covariance matrices have two key features: symmetry and positive definiteness[27], both of which are crucial for the filter's stability. In a computer implementation of the (E)KF both of these features may be lost due to rounding errors. According to [Bar-Shalom and Li 1993] there are two main points of trouble (with the plain KF, but for the EKF the situation is similar):

1. The covariance prediction step

   $$P_{t+1}^{-} = F P_t^{+} F^T + Q$$

   as in equation 3.15 can affect only the symmetry of the resulting matrix, a problem which can be avoided with a proper implementation of the matrix multiplication.

2. The covariance update step

   $$P_t^{+} = (I - K_t H)\, P_t^{-}$$

   as in equation 3.18 can affect the symmetry and – because of the subtraction – the positive definiteness. However, algebraically equivalent forms exist that do not have this problem.

Another issue is that of *efficiency*. The covariance matrix has a size of $n \times n$, where n is the size of the state. For a mobile platform with coordinates of a lot of landmarks in the state this n can be quite large, leading to very large matrices. On the other hand side these matrices are likely to be only sparsely filled, so a naive approach is likely to waste memory and computing time. In case this should become a problem there are special software libraries available for sparse matrices.

[25] `http://www.mysteryspot.com`

[26] Note that stability of the filter does not require the dynamic system to be stable. Filter stability means that the covariance matrix converges to a finite steady (stable) state, whereas state models used in tracking are unstable because they have an integration from velocity to position. Stability means "bounded input bounded output", and this condition is not satisfied by an integrator [Bar-Shalom and Li 1993].

[27] An only positive *semi*-definite covariance matrix, i.e. one with zero eigenvalues for some state components would reflect the filters belief that it has perfectly accurate estimates of these state components – a thing which is unlikely to be true and therefore undesirable.

3.2.8.4 Consequences

As a final word it can be said that the EKF is a suitable approach for the localisation problem if the necessary precautions are met. In such a case it is in particular quite computationally cheap, compared to other solutions. For example the *particle filter* spends more effort on exactly modelling the error distribution and therefore being able to better choose the likeliest solution, but at the price of increased computational effort[28]. This can very well be a knock-out argument for highly integrated robots where computing power is typically low compared to common desktop computers.

3.2.9 Realisation

The control flow for the localisation is separated into three major threads, as can be seen in figure 3.22. One thread reads the laser scanners and calls its `update()` function once it has received scans from all (both) scanners, one thread reads the odometry and gyro values and calls its `update()` function once it has received all information and one thread periodically calls the `predict()` function and executes motion steps. This way the asynchronicity of the different sensors (see again figure 3.20) is completely hidden from the EKF.

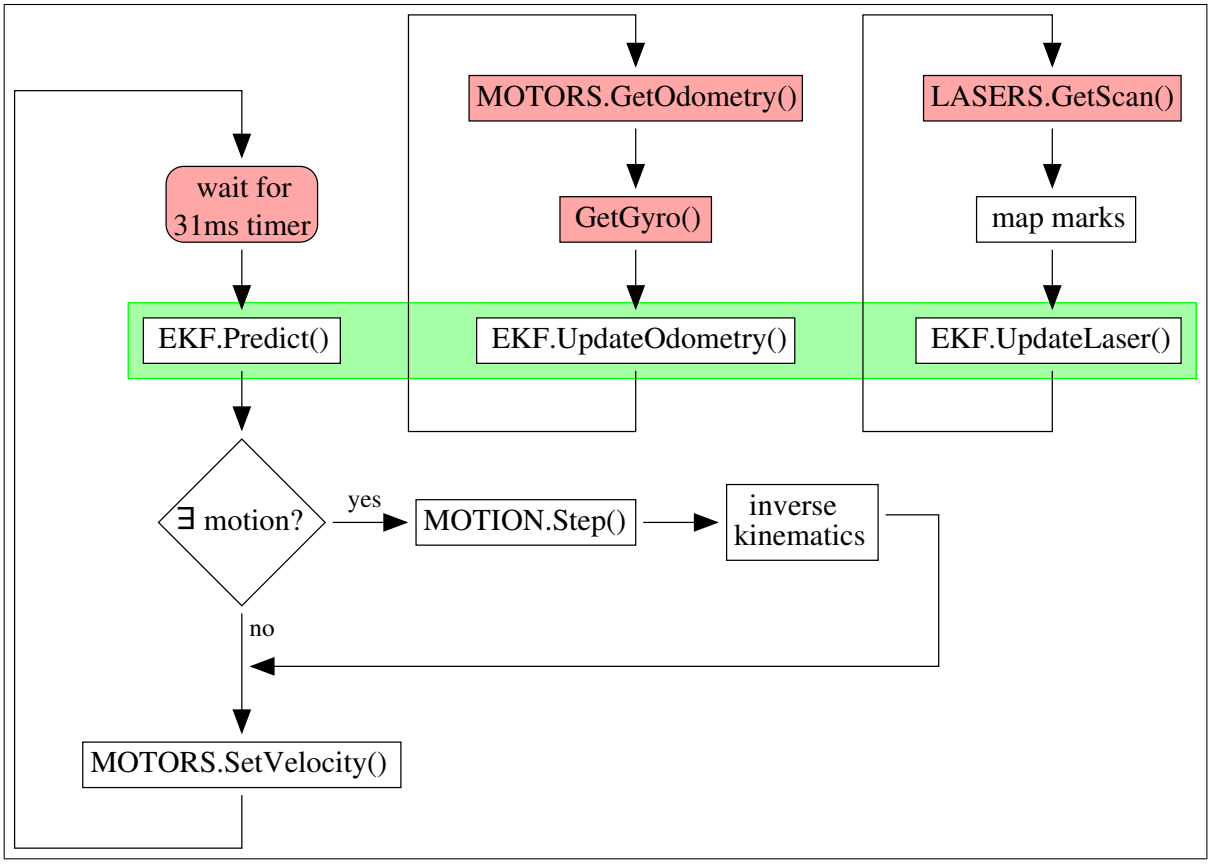


Figure 3.22: Control flow for the localisation. Boxes underlayed with red are actions that can block for a longer amount of time and are therefore executed by parallel threads. Boxes underlayed with green belong to the EKF.

[28] See [Wright, Maskell and Briers 2003].

3.3 Path-Planning

Path-planning is the task of computing an optimal path from an arbitrary start point to an arbitrary target that avoids collisions with known static objects (obstacles). This involves aspects of maps (which are needed to store knowledge about the obstacles), graphs (which are needed to express paths in these maps) and the actual path-planning (which means searching in these graphs) – topics which are discussed in this section.

Searching a path means to minimise a *cost* function. These costs can be expressed in many ways, for example in terms of lengths of distances travelled, lengths of time intervals needed and amount of electrical energy needed. For this work the cost has been defined as the length of the path, the shortest path therefore being the optimal one. Since planning a collision-free path means to go around obstacles a *graph* of all possible straight connections between corners of obstacles that keep a certain safety distance is generated from the map and labelled with the corresponding cost. For problems like this, where from any point of the graph the minimal remaining cost to the goal can be estimated without evaluating the graph, the A^* algorithm can be used to optimise the search. The resulting path is the shortest sequence of straight lines that lead to the goal without coming closer to any obstacle than the specified safety distance.

The *obstacle avoidance* is at this stage limited to known static objects because only they are known in advance. Dynamic objects like humans roaming around the scene can not be handled here because their positions can not be forecast and stored in the map. For simplicity and because the given problem description does not necessarily require it the solution presented here does not go as far as maintaining models of dynamically moving objects and therefore having possibly dynamic paths. Dynamic obstacles will be taken care of at the motion execution stage in section 3.4, where they are dealt with by a *collision avoidance* module.

Since the field of mobile robots is a quickly growing area of scientific work there are of course a lot of other ideas and approaches which have not been incorporated in this work. The reason for this is that most of these alternative ideas are targeted at problems which do not occur at our setup or do at least not have a high priority. Amongst these ideas are – to mention only the most commonly known ones – *simultaneous localisation and map-building* (SLAM), which was not included because the setup requires all positions to be known in advance and the map-building process is not that difficult for a single laboratory. A second idea are *behaviour* based approaches, which were not included because they can fatally interfere with the static path-planning on the one hand side and can not be guaranteed to reach a target themselves on the other hand side. They can unboundedly violate the optimality criterion up to the point of causing more collisions in the course of avoiding one in the first place – a loop which can not be guaranteed to terminate.

Again, this section is a compressed version of the work Schneider and Westhoff have done in and for this project[29].

3.3.1 Maps

The necessary prerequisite for every path-planning is a proper map, representing the robot's knowledge about its environment. Although such a map may appear trivial at first sight, it is not. First there are several types of maps depending on what the robot needs to know about its

[29] See [Schneider and Westhoff 2002], chapter 6.

environment, then there are several ways of storing information about obstacles in the maps and finally the object which is to be moved according to the map may have an influence on the map design.

This section introduces the questions of *map design* with the given setup of the laboratory in mind. It shows why an appearingly primitive polygonal map has been chosen and how this map is modified during *map expansion* to take the shape of the robot into account.

3.3.1.1 Map Design

Maps can be divided into different groups according to how much information they hold and of what type that information is. Gallistel has defined a geometric concept of "strength", allowing to compare different map representations. He says that the "strength" is

> *[...] the range of geometric relations among the mapped points that could in principle be recovered from the map [...]*[30]

Lee in turns has used this "strength" to set up the following four basic categories of map types[31]:

- **Recognisable Locations:** The map consists of a list of locations which can be reliably recognised by the robot. The recognition is only a qualitative but not a quantitative one, which means that no exact values for the positions are available.
- **Topological Map:** In additions to the recognisable locations, the map records which locations are connected by traversable paths.
- **Metric Topological Map:** This term is used for maps in which distance and angle information is added to the path descriptions.
- **Full Metric Map:** Object locations are specified in a fixed coordinate system. Precise positional information is provided.

A good example for recognisable locations are vision systems which classify images into known categories, like learning based *omnivision* systems[32]. These systems yield an abstract similarity of images to learned situations, from which at most a rough position guess of an accuracy of at least several centimetres can be derived. They may therefore be usable for a coarse monitoring of the (E)KF, but not for building maps where a high precision is needed.

What the given setup requires is that positions are precisely known in order to allow planning proper paths around obstacles without collisions and detecting and compensating for malpositioning of target objects. This requires a full metric map, and in fact the localisation has already been designed to be able to provide the necessary precision. Also, any map with less strength than a full metric map can not explicitly hold obstacles – they are only stored implicitly by the paths.

The obstacles themselves are then best and easiest described by their polygonal outline, resulting in a *polygonal map* as in figure 3.23 (the objects only being filled for visualisation). Such a map is also suited for the *map expansion* that is needed to implement the obstacle avoidance.

[30] See [Gallistel 1990].
[31] See [Lee 1996], page 18.
[32] See [Zhang, Hübner and Knoll 2001].

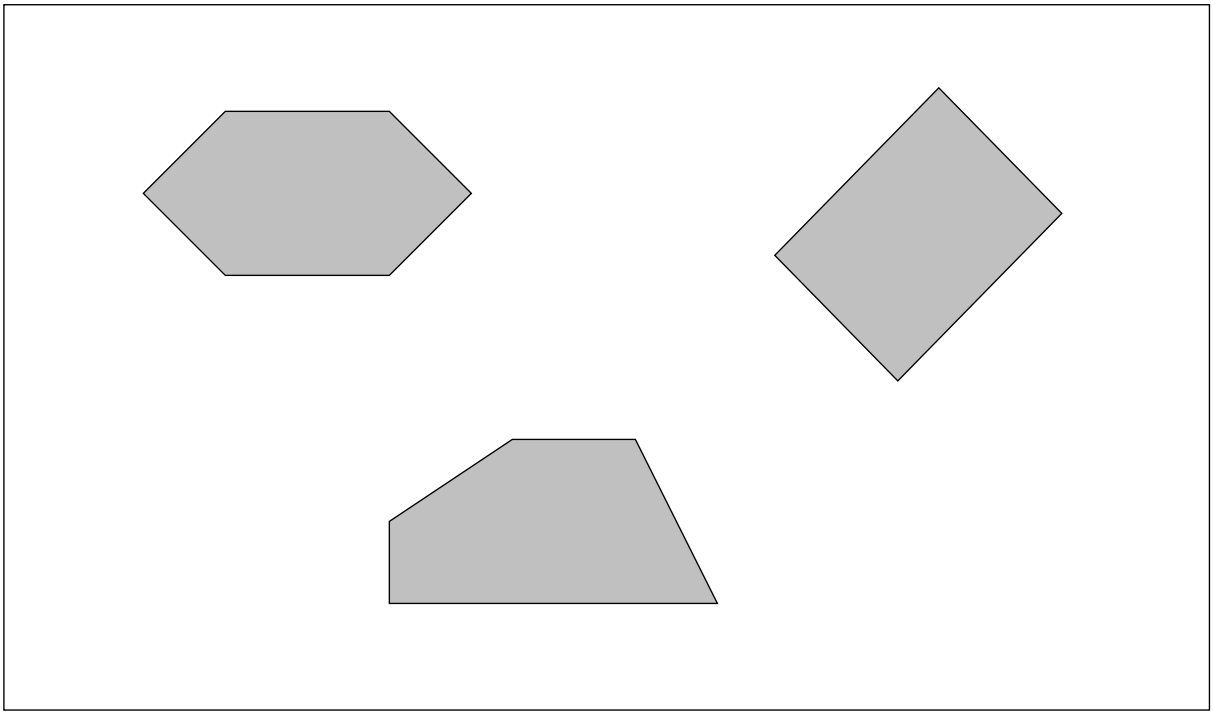


Figure 3.23: A polygonal map of obstacles, which are only filled for visualisation.

3.3.1.2 Map Expansion

In this polygonal map the robot has to move, which raises the question of its shape. Obviously it makes a difference whether an arbitrarily shaped object has to be moved or some constraints about the shape can be made [33]. Arbitrary shapes will not only need translations to get to the goal, but also rotations to get around obstacles.

The more rotationally symmetric the shape of the robot is the less rotations are needed and the easier the problem is, down to a fully rotationally symmetric circle which will not need any rotations. The most easiest case would be if a *point* has to be moved in the map. Fortunately the shape of the MP-L655 can as a first approximation be considered circular without introducing too much error, which allows to use the idea of *map expansion* that makes it possible to reduce the platform to a point.

Expanding Lines / Shrinking Robots

Given a robot radius of r, the basic idea behind map expansion is to shrink the circular robot to a point and at the same time expand the obstacles by r. Applied to a single line as in figure 3.24 this means to build parallels to the line at a distance of r and to build semi-circles with a radius of r at the line's endpoints. The expansion yields a "cigar" shape around the line. The original obstacle avoidance goal of *not coming too close to a line* then translates into *not crossing the outline of a cigar*.

Although the semi-circles can be stored with only four parameters (the position x and y, the radius r and the orientation α) using them is not an optimal idea. The reason for this

[33] This is known as the *piano mover problem*, see `http://www.google.de/search?q=piano+mover+problem` or `http://www.cs.caltech.edu/~lei/cs20/c/project/src/pm.html`.

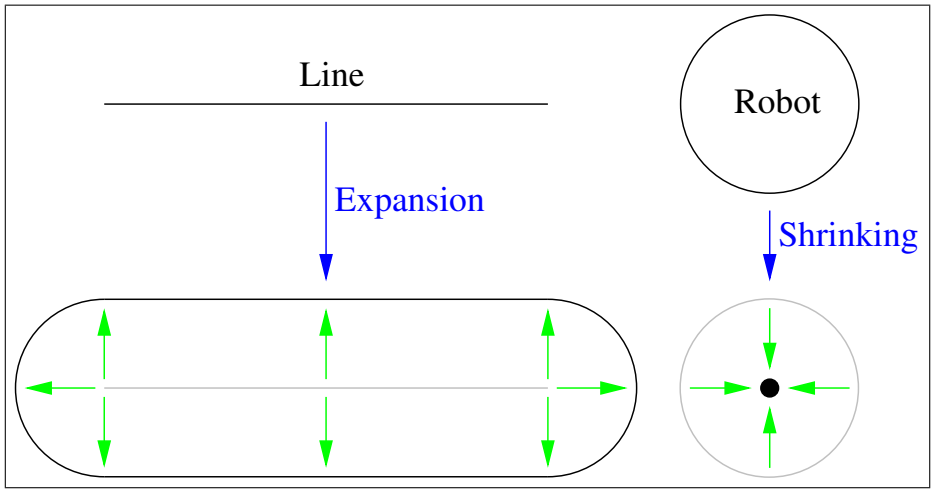


Figure 3.24: One idea of moving a robot with a radius r through a map is to shrink the robot to a point while expanding the obstacles by r. Applied to a single line this means to build parallels to the line at a distance of r and to build semi-circles with a radius of r around the line's endpoints.

is that the path to be constructed is likely to follow only fragments of these circles, which means that intersections between circles and lines will have to be computed. These intersections are significantly more complex compared to intersections of lines only. In addition to that, intersecting (nearly tangential) lines with circles is computationally troublesome because it is much more affected by rounding errors than intersecting two lines.

In order to keep the computations stable the circles can be approximated by polygons as in figure 3.25. For this work each semi-circles has been approximated by 3 lines, which increases the amount of information stored for each original line from 2 lines plus 2 semicircles to 8 lines total.

Taking into account that a line is also stored with four parameters (the start point x_0 and y_0 and the end point x_e and y_e) this means that the number of parameters stored for each original line increases from $(2+2)\cdot 4 = 16$ to $(2+2\cdot 3)\cdot 4 = 32$ total. This is a significant increase of information, but the gain in simplicity of the intersection computation is well worth the price.

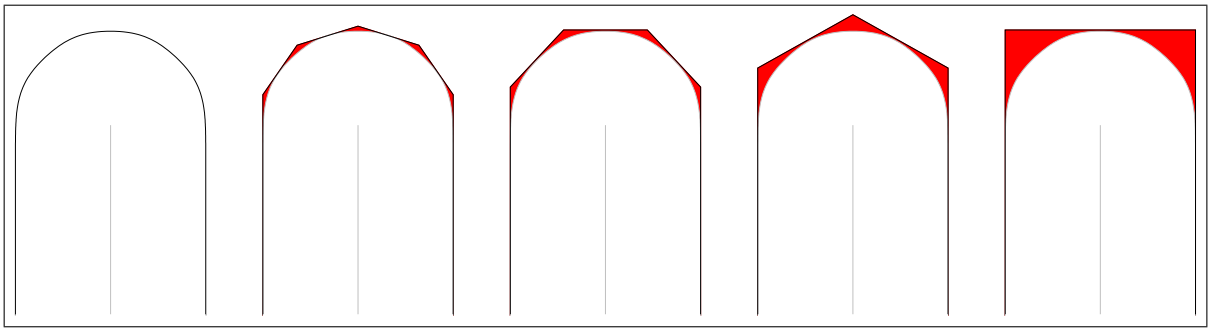


Figure 3.25: The circles around the endpoints of a line can be approximated by a polygon with more or less lines, depending on a tradeoff between accuracy and modelling complexity. From right to left: 1, 2, 3, 4 and ∞ lines. The area marked in red indicates the excess space that is wasted.

The only potential problem that remains is that the approximation blocks more space than the circle would have done, but with 3 lines per circle this can usually be neglected. The only case in which this might become a problem are very narrow passages, but since these are already potentially dangerous adding a bit more safety distance does not introduce a new disadvantage and can only relax the problem. If this means that the passage will get blocked the situation is so dangerous that it should better be re-arranged in the first place.

Expanding Objects

The above mentioned expansion method is designed for expanding single lines only, and not composite objects. Instead of describing the world with single lines only a more sophisticated approach is therefore desirable. Objects – for the purpose of obstacle avoidance – can be fully described by their 2d-shape, their contour, which in turns can be approximated by a polygon. Since a polygon is only a sequence of lines the easiest way to expand an object is to built parallels to each line and join neighboured ones, and the easiest way to do that is to use simple connecting lines.

As can be seen in figure 3.26 in case of concave corners of a polygon the parallels intersect before the joining line. This is not yet a real problem, but only a waste which can in many cases be avoided by not joining the parallels but using the intersection point to cut them off. An exception are situations where the lines are so short that the parallels do not intersect. This problem will soon be addressed.

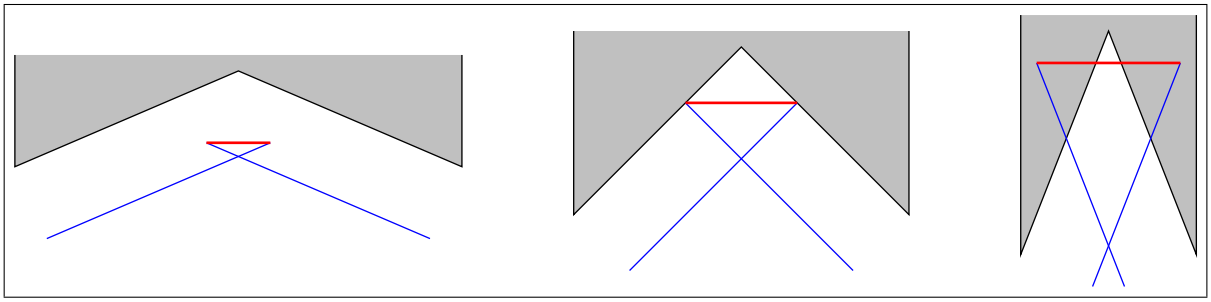


Figure 3.26: Problems when joining parallels at concave corners.

The opposite case of convex corners also yields problems, as can be seen in figure 3.27. To keep the desired distance the parallels may or may not be joined by connecting lines, depending on the angle with which the original lines meet. If the angle is too pointed a simple connecting line will violate the distance constraint[34]. It is possible to use more complex construct like arcs to bridge the gap join the parallels, but that only works if the lines are long enough or no concave corner follows.

One implementation of such a simple approach is *edge shifting*[35]. In edge shifting the parallels (edges) are built as above, but allowed to extend beyond their original length. As suggested for concave corners above, the intersection of the extended shifted edges of two neighboured lines is taken as a corner point of the expanded contour. This process is repeated for each neighboured pair of two original lines until a complete expanded contour is obtained.

[34] More precisely any such line will violate that constraint because the end point of the parallels are on a circle with the desired distance radius around that corner, and so a line between them is a *secant* to that circle. But in case the distance was amply chosen this can be tolerated to some point.

[35] See [Hunn 1993], p. 18-22.

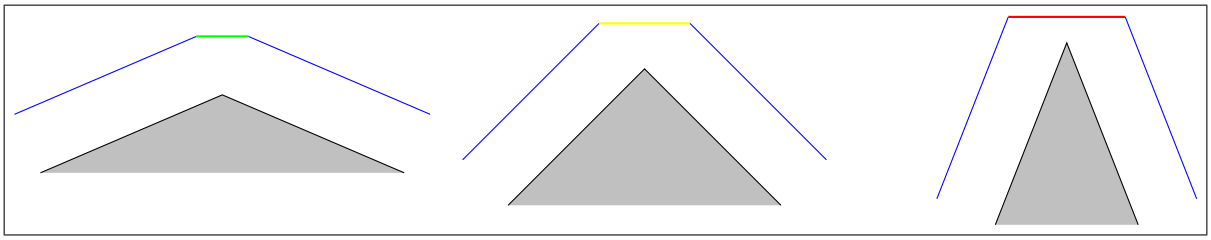


Figure 3.27: Problems when joining parallels at convex corners.

This basic version of edge shifting is actually not very much more sophisticated than the joining with simple lines, and it is therefore not surprising that it suffers from similar problems. As can be seen in figure 3.28 it does neither guarantee to get the optimal contour, nor even legal ones:

- In the left situation the expanded contour reaches into the original object. This is again not a real problem, but only a "suboptimal" contour. The robot can not collide with the object because it would have to cross a line in order to get close enough to it. However, the "loop" in the expanded contour unnecessarily increases the number of lines to check, and therefore the required processing time. In order to reduce that time to a minimum the loop could be removed, but this process is again not at all trivial as it involves not only neighboured lines, but the whole contour.

- In the right situation the problem is more severe. Due to the shape of the original contour the extended shifted edges intersect at corners that are off the optimal expanded contour. In some cases they "only" waste space, but in other – fatal – cases the robot can collide with the object. The reason for this is that as with removing the loop looking at only two neighboured lines for each corner is not enough, because it might lead to a collision with a third line. Again, to get around this, *all* lines have to be looked at, which significantly complicates the expansion process.

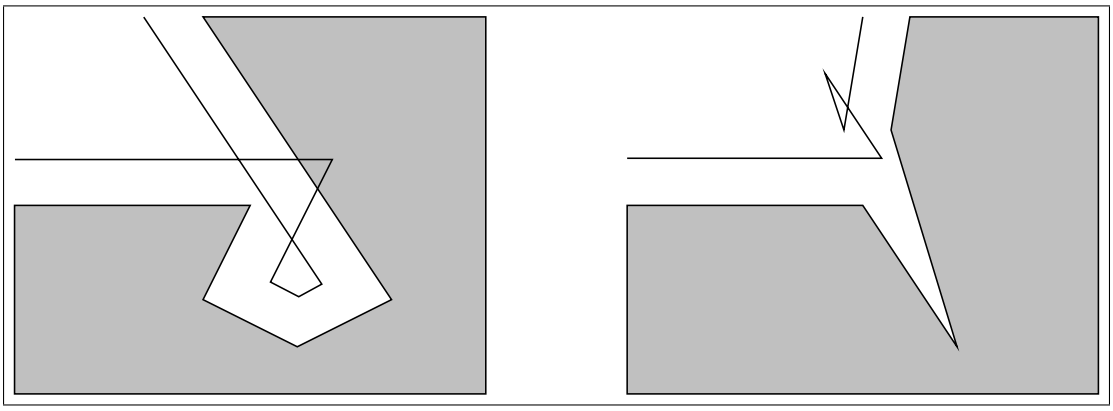


Figure 3.28: Problems of the *edge shifting* approach: The expanded contour may be suboptimal and even reach into the object (left) or even allow collision with the object (right).

The problem of wasting space can be seen in figure 3.29: At very sharp edges the intersection of the shifted lines lie much farther away from the object than necessary. This not only blocks more space around obstacles than necessary, but can actually lead to that no path is found because a passage may be blocked. Again, avoiding this situation means to use other

means of joining the parallels, means which are likely to cause problems or at least an increased complexity at neighbouring corners.

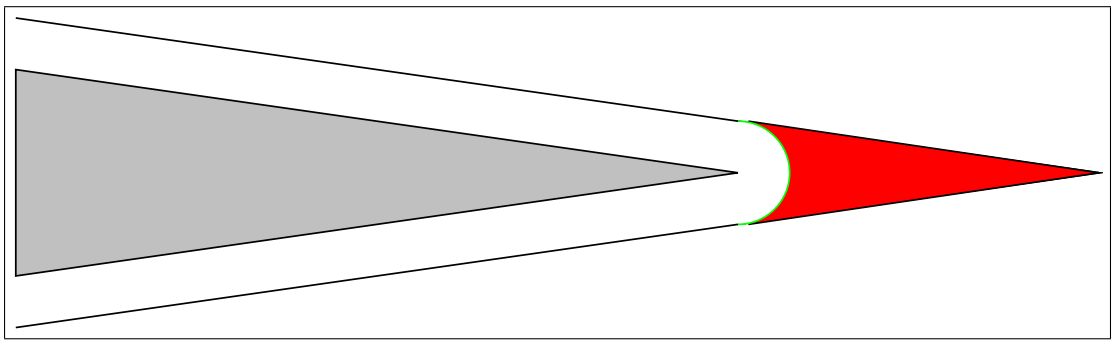


Figure 3.29: The edge shifting approach blocks more space around obstacles than necessary.

Adding to all these problem, edge shifting only works for *solid* objects given by *closed* polygons. The reason for this is that it does not specify how to join the "inner" and "outer" parallel at the endpoint of an open polygon. Only at closed polygons this situation never occurs. This is an undesirable restriction as it often suffices to use plain lines or open polygons to model obstacles for the robot. Also, adding obstacles may require recalculation of the expanded contour over much larger areas of the map than just the newly added part, and using only local sections of the map to optimise the processing time is much more difficult as it may cut closed polygons into open ones. Because of these difficulties approaches trying to compute an "optimised" expanded contour have not been regarded for this work.

Instead, in this work the obstacles are allowed to consist of open polygons (including single lines) and each of their lines is expanded separately using the cigar expansion[36]. Figure 3.30 gives an example of how this works for a simple polygonal object. It can be seen that this approach yields a quite large number of lines, many of which are redundant because they lie completely inside the cigars and will therefore never be touched.

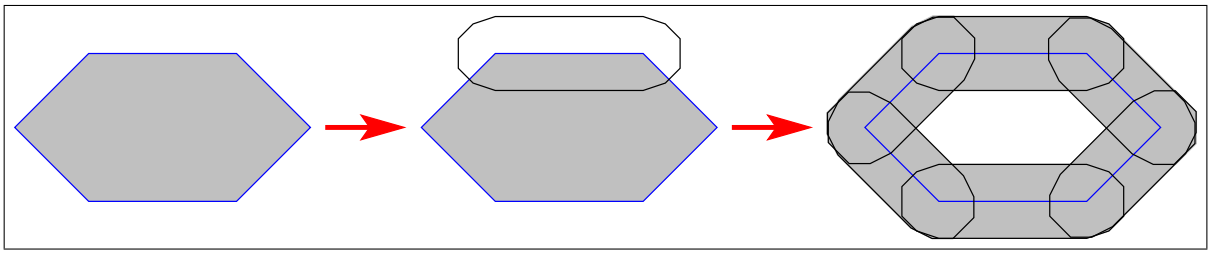


Figure 3.30: The cigar expansion – when applied to polygonal contours of objects – leads to a large number of lines which are redundant because they lie inside the expanded contour.

The redundant lines can optionally be removed by applying the following checks: For each line of each cigar it has to be checked if both its start and end point are within any of the other cigars, which is very easy to do because the cigars are convex polygons[37]. If they are, the line is tagged. After this process has been completed all tagged lines can safely be removed from the

[36] Actually the polygons do not really exist. They are only used as a theoretical construct to allow the writer of the map to have an idea of "what belongs together". Internally all lines of all polygons are stored individually.

[37] See [Sedgewick 2001].

object, which will cut open the closed cigars. The remaining cigar fragments are open polygons representing the object's shape without forming closed contours, as can be seen in figure 3.31.

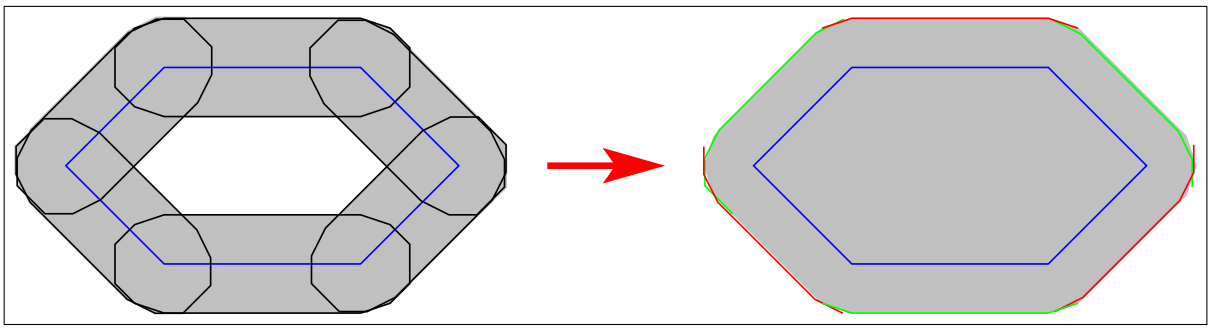


Figure 3.31: Deleting the redundant lines, whose start or end point in in a cigar. For better visualisation the segments resulting from the different cigars have been marked in red and green.

A next step might be to compute the real closed polygon of the object's shape. The set of open polygons resulting from the above step can be merged at their intersection points to obtain a single closed polygon. This does not affect the computational effort as it does not change the number of lines, but it allows the notion of being "inside an object". However, this idea of reducing the object to its outer hull destroys an interesting feature of the cigar expansion approach. It has to be repeated that the initial goal of

Do not come too close to an obstacle.

has in the course of expanding the map already been modified to

Do not cross any lines in the (expanded) map.

More precisely, the path-planning task imposes the following two restrictions on a legal path:

1. The start and goal point must not lie inside a cigar.
2. The path between them must not cross any line of any cigar.

Note that this does not explicitly say that the start or goal points may not lie inside an object, simply because the notion of "inside an object" would require the notion of an "object" in the first place. Since in the simple cigar expansion these do not exist the "objects" – if large enough – will have a free space inside them (see again figure 3.30) which does not violate the first of the above constraints. Attempting to build a path from such a point to the "outside" of the object will violate the second constraint, but attempting to build a path to a point which is also "inside" the object does not. This makes it perfectly legal for the robot to apparently move inside an object.

The practical upshot of this is that for example a table can be modelled by giving its contour and a small robot can then be placed and safely moved on it without falling off the edge. The question of whether this is applicable to an "object" or not is reduced to whether or not it is possible to legally put the robot there in the first place. This side effect is only possible because there is no notion of the "inside" or "outside" of an object, or even of "objects" – there is only the notion of "crossing lines".

As a result of these consideration the cigar expansion for single lines has been chosen for this work. The redundancies of lines is not addressed here, but dealt with at a later stage. The resulting expanded maps are therefore actually more complex than shown in figure 3.32, where only the outer contour is given and the obstacles appear to be solid. These simplification have only been applied to this and the following figures to yield more sparse and therefore better visualisable graphs.

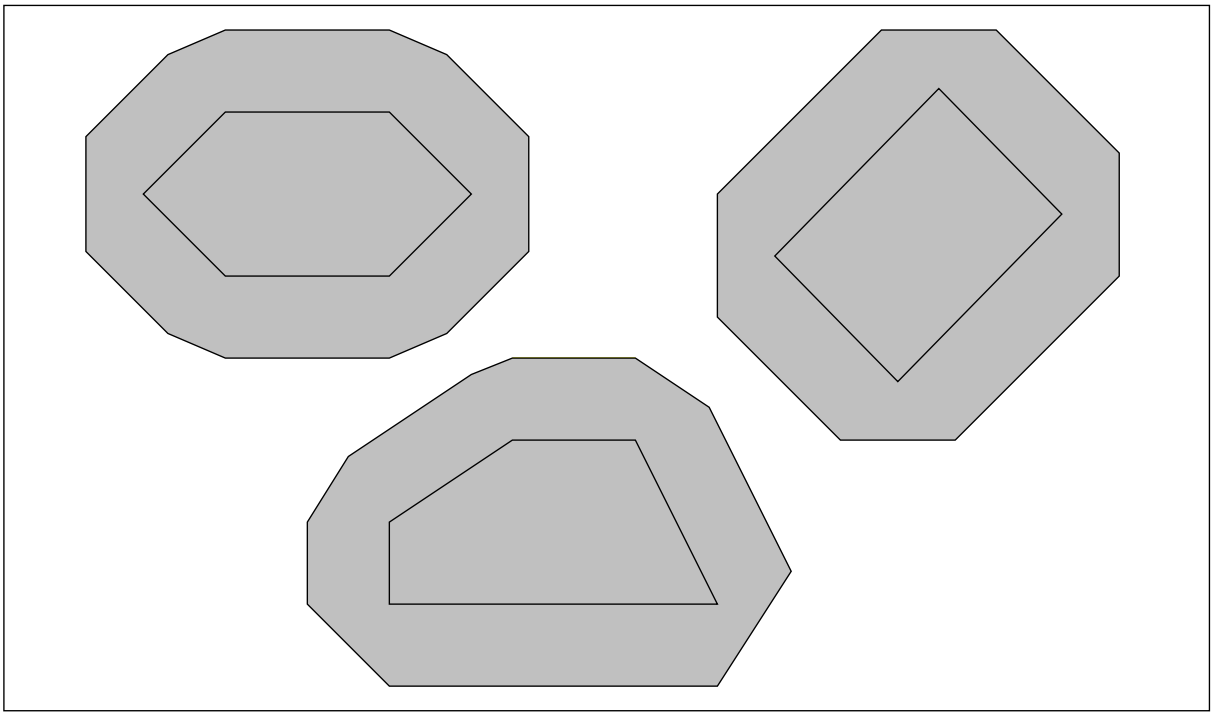


Figure 3.32: The expanded polygonal map. Note that the shifted edges are directly connected to form the expanded contour instead of using the cigar expansion. This violates the distance constraint but keeps this and the following figures simpler. For the same reason only the outer contour is given and the objects are filled to appear solid.

3.3.2 Graphs

The expanded map is only the foundation for the remaining steps of planning a path for a motion. The start and goal point of the motion are still unknown, but nevertheless a lot of information about admissible paths can already be extracted from the map at this stage, so that the upcoming path search is as optimal as possible.

Having reduced the robot to a point the task of searching the shortest path is one of moving a point along lines. The best way to store information about points and lines is a *graph*, because there exists a vast number of algorithms for operating on graphs. In the following the idea of visibility and tangent graphs for describing admissible and optimal paths will be presented, which finally leads to the A^* algorithm used for the path search.

3.3.2.1 Visibility Graph

In a world that consists entirely of lines, intersections of two lines – or, in other words, the corners of obstacles – are the only prominent and interesting points. Except from the part

from start or goal to an obstacle the optimal path can only consist of lines between corners of obstacles. This is because any shortest path will consist of a segment from the start to a corner of an obstacle, some segments along the contour of that obstacle and/or to corners of other obstacles and finally a segment to the goal. Any deviation off the contour will make the way longer. This includes the one special case where the goal is directly reachable from the target.

To built a foundation for the path search a *visibility graph* (VG) is constructed from the expanded map. The graph is initialised to hold all corners from the expanded obstacles as nodes, and no connections. Then, for each corner a connection to all those corners that are visible from it is added to the graph. Technically this is implemented by a *matrix* whose elements are either the positive cost of the connection of the corners denoted by the row and column indices or negative if no legal connection exists. An example for such a VG can be seen in figure 3.33.

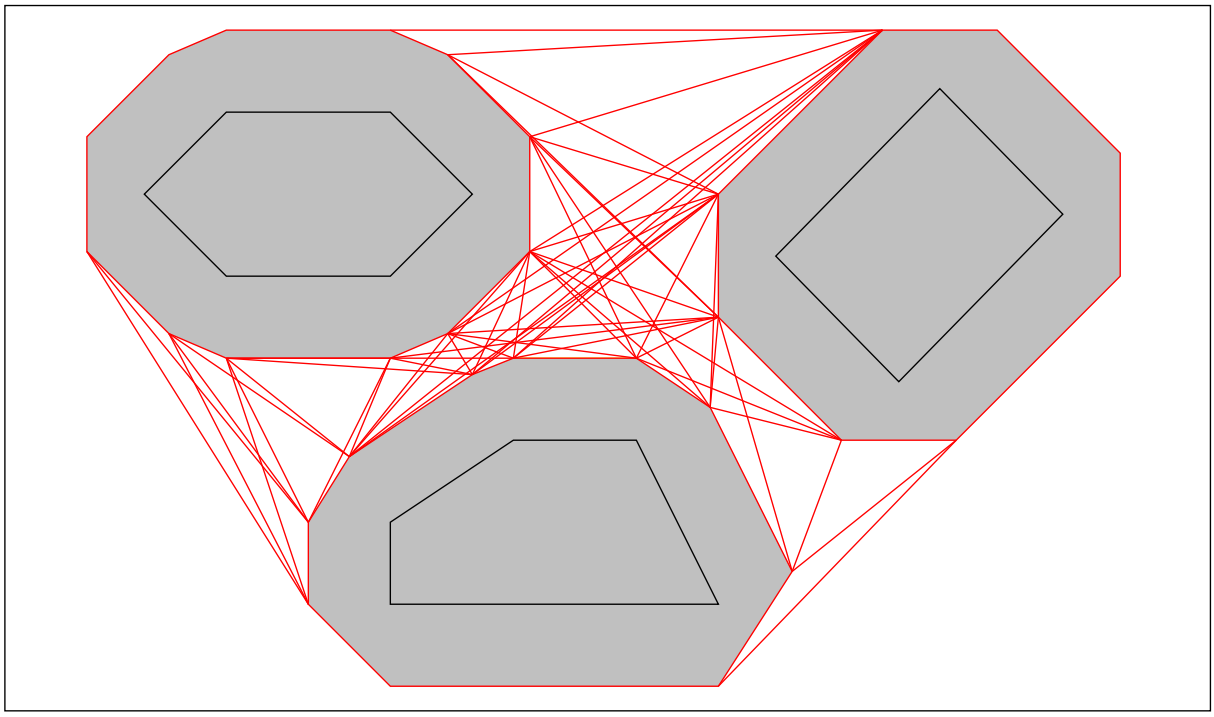


Figure 3.33: The *visibility graph* (VG), based on the expanded polygonal map. For each corner there is a path to each other corner that is visible from it. This automatically includes its neighbours in the contour, making the complete contour a possible path.

This figure is again a simplification as it is not based on the real cigar expansion. Since it only uses the outer contour it has a quite trivial and obvious notion of visibility. Visibility between corners in the cigar expansion context means that

1. the start and goal corner must not lie in any cigar and
2. the line between them must not intersect any cigar.

Using these constraints yields a graph with actually much more connections than in figure 3.33, for example connections between corners in the free space inside objects (see again figure 3.30). It is the second constraint that is important because it forbids connections through cigars and therefore keeps the robot from coming too close to any obstacle.

Since the run-time of any search algorithm depends on the number of points to visit the resulting graph is not yet an optimal base for path searching – it holds too many connections. The next step therefore is to reduce the VG by deleting connections that are redundant.

3.3.2.2 Tangent Graph

As has been suggested a path will consist entirely of segments between corners of contours (with the exception of the start and goal point). More specifically, once one obstacle is passed the shortest possible path around the next obstacle is to go to the farmost visible point, the one where the line only touches the contour in a single point. Any attempt to first go on the contour and then around it will only make the path longer (with the exception that the target lies on the contour).

Therefore all those connections between corners where the prolongation reaches inside the contour are deleted from the graph. The resulting graph only holds connections which are tangential to both their contours, and is therefore called a *tangent graph* (TG). As can be seen in figure 3.34 the TG holds significantly fewer connections than the VG it was constructed from.

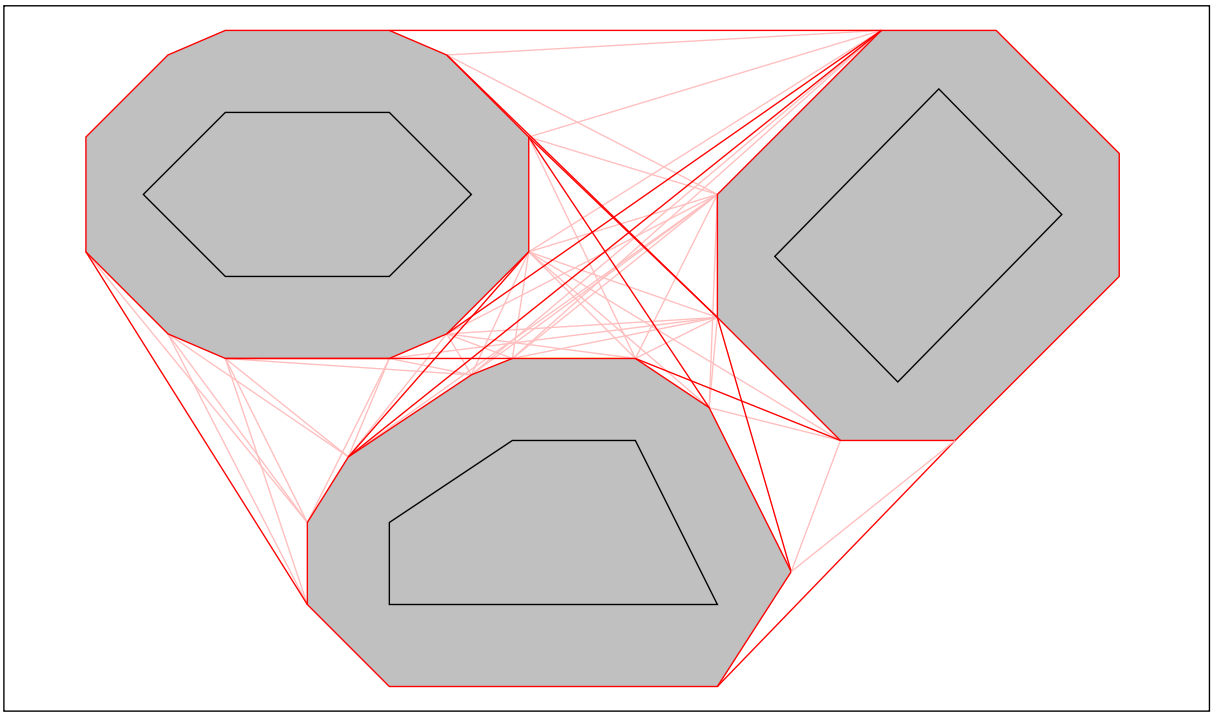


Figure 3.34: The *tangent graph* (TG), based on the visibility graph. All those paths are discarded which are not tangents to the obstacles they end at, i.e. whose prolongation would reach inside any of these obstacles. This results in a significant reduction of the number of paths.

The tangentiality attribute has to be computed with caution because erroneous deletion of a connection may lead to making parts of the map unreachable. In this work an algorithm based on the `ccw()` function by Robert Sedgewick[38] is used. This algorithm basically computes whether a point is on the right or left side of a line. Tangentiality can be computed from this because if a line is a tangent to a contour point, then the point preceeding and the point following that point on the contour have to be on the same side of the line.

[38] See [Sedgewick 2001].

This computation is sensitive against rounding errors as is illustrated in figure 3.35. If both of the connections (p_3, p_5) and (p_2, p_5) are deleted no passage through the gap between the objects will be possible. Such a situation can only happen if the connection in question is (almost) parallel to a contour line. One way to avoid this problem is to modify the `ccw()` function to return the real distance of a point to a line and to use a threshold that is knowingly larger than the worst possible rounding error.

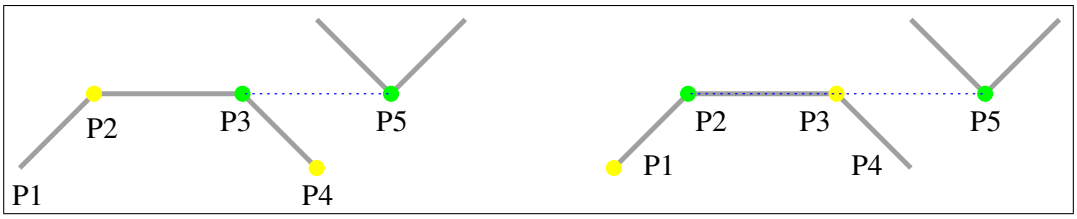


Figure 3.35: Sensitivity of tangentiality computation against rounding errors. If in the left case the point p_2 is erroneously taken to be above the line (p_3, p_5) it is on the opposite side than point p_4 and (p_3, p_5) can not be a tangent. If at the same time in the right case (p_2, p_5) is also not accepted as a tangent because p_3 is erroneously taken to be above that line no connection between the two object remains.

3.3.3 Path Search

On the above presented TG the path must be searched on, but before a path can be searched the start and goal point must be known, which leads to a final modification of the graph. Once they are added the A^* algorithm can be used to search the optimal (shortest) path.

3.3.3.1 Arbitrary Start and Goal Points

So far the TG contains only path fragments concerning the corners of (expanded) obstacles and not yet the current position or goal. This is because the obstacles represent the only information that is available off-line for advance computation, and advance computation is desired to keep the on-line computation effort low. Only when a motion is to be done and the start and goal point are known they are added to the graph on-line. All connections from the start and goal to the directly visible corners that are tangents are added to the graph, finally resulting in the *path graph* (PG) as in 3.36.

Since this step has to be repeated for every motion its computational complexity has to be analysed: Adding two points (corners) to the TG increases the size of the vector of nodes by two and the size of the matrix of connections by two rows and columns. For any number n of original corners the size of the matrix will increase from n^2 to $(n+2)^2$. The increment in size $(n+2)^2 - n^2 = 4n + 4$ means that the effort to add the start and goal point will increase with the number of points that are already in the graph, but only linearly. It is therefore likely to be neglectable.

If, in case of very large graphs, this should become a problem there are still a number of alternatives possible. If the number of possible start and goal points is limited (for example the devices in the biotechnological laboratory) and only the number of obstacles is high (for example because the devices are spread in a very large room with many other objects) all start and goal points can be added to the TG right away and the path(s) pre-computed. In this case

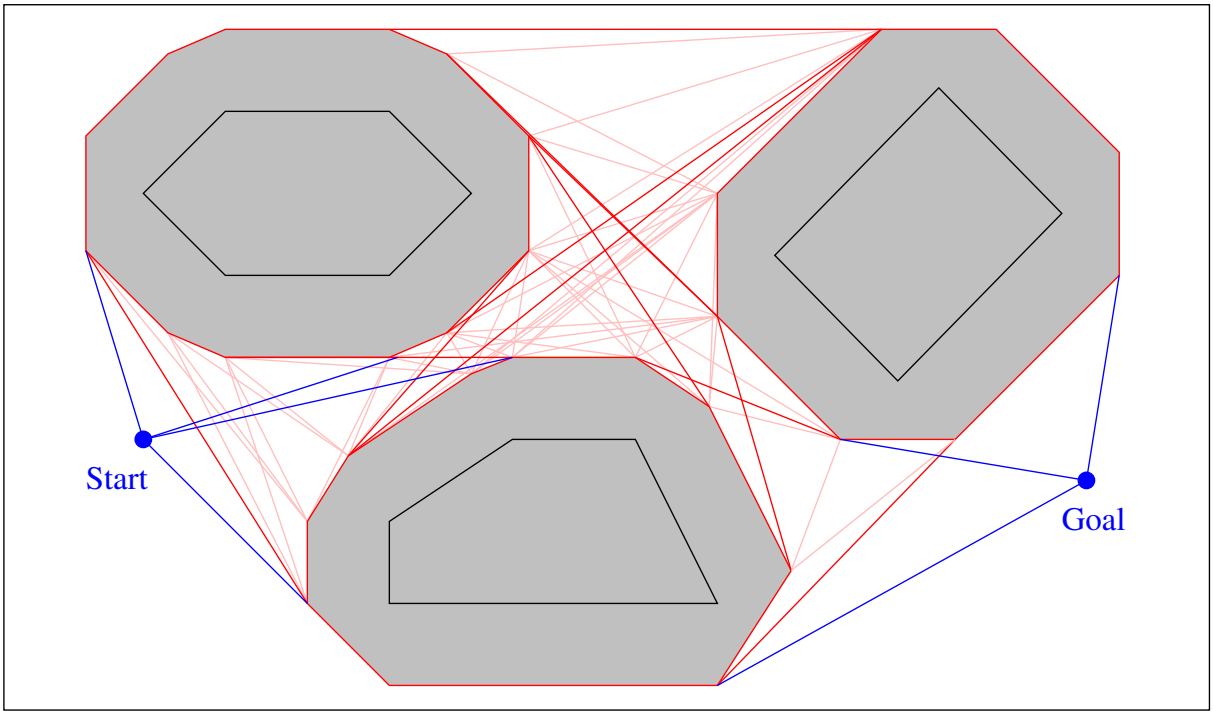


Figure 3.36: The start and goal point are added to the tangent graph by adding (blue) paths from them to any corner which they only touch tangentially, yielding the *path graph* (PG). This operation typically adds only comparatively few connections.

the only thing that has to be checked at run-time is whether the robot really is at one of the start points, or can at least directly reach one of them. Since this is under normal circumstances very likely to be true the complete stage of path-planning can then be omitted.

Another possibility, especially if multiple rooms are involved, is to split the map and the graph into smaller parts with defined juncture points. This way only those partial graphs holding the start and goal point have to be modified, leading to a reduction of required computing time.

Also, it might be worth considering to use a control mode which already puts the robot into motion as soon as parts of the path are known and does not wait for the complete path. Even though it is not guaranteed that the first available path segment guess is really the optimal one it is still very likely and can speed up the motion execution. Parallelising the path-planning and the motion execution also means that the path-planning does no longer have to be done on the rather small embedded computer on the platform, but may be passed over to a more powerful computer in the network. Such a centralised PC can then also partly synchronise the motions of two or more robots which would otherwise not know about each other.

3.3.3.2 A^* Algorithm

After inserting the start and goal point the TG contains all the information that is needed to finally search the path. Searching basically means to try several possibilities and choose the "best" of them, according to a certain cost function. In case of a geometric problem like path-planning this cost function is most commonly the length of the path, but other cost functions are also possible (like the smoothness of the path or the energy needed to drive it). Graphs are a very well researched field of computer science and so a lot of algorithms exist.

The obvious possibility is to begin at the start point and try all possible connections to other points. This yields a set of paths which consist of only one segment yet, each with a different length (cost). No assumption about which of the paths may be preferred is made, but instead again all possible connections to other points are tried (except from those leading back on the path). This leads to a set of paths which consist of two segments. The process is repeated until all combinations of segments have been evaluated, and from the (hopefully non-empty) set of solutions the best one is chosen.

This *breadth first* approach is known as *Dijkstra's algorithm*. It does find the best solution, but at the price of completely evaluating all possibilities. It does not take any information into account other than the length of a complete path. It is therefore also called an *uninformed* search.

The total cost $c(i)$ of a path i is of course only known if the goal $\vec{g}$ is reached, but if an estimate $e(i)$ about the remaining costs would exist the total cost $h(i)$ of a partial path could be estimated as

$$\boxed{h(i) = c(i) + e(i)} \tag{3.29}$$

This estimate can be used to improve Dijkstra's algorithm by evaluating the best path first. A path with higher estimated remaining costs is considered unlikely to turn out to be shorter than one with lower estimated remaining costs. The first optimisation is therefore that is is likely that the first solution is found in less time. This solution is not guaranteed to be the optimal solution and so the evaluation has to be continued, but the second optimisation is that some of the remaining partial path(s) j can be completely discarded because their estimated costs $h(j)$ are higher than the total cost $c(i)$ of the so far best solution. The only condition required for this to be true is that the estimated remaining cost $e(i)$ in equation 3.29 does **not** over-estimate the real remaining cost, and for geometrical problems like looking for the shortest path the easily computable euclidian distance

$$\boxed{e(i) = \|\vec{p}_i - \vec{g}\|}$$

between the the current end point $\vec{p}_i$ of the partial path i and the goal $\vec{g}$ guarantees exactly that.

This *best first* approach is known as the A^* algorithm by Hart, Nilsson and Raphael[39]. It yields the optimal path computed in an optimal time. An example of the result of the A^* computation can be seen in figure 3.37, where the best solution has been marked in green.

3.3.4 Summary

The path-planning part of the mobile platform control for static obstacles is implemented using the A^* algorithm on tangent graphs, which are in turns computed from visibility graphs. These graphs are derived from line maps in which each line – representing a part of a (not necessarily closed) polygonal obstacle – is expanded by the robot's radius r using a "cigar" expansion, while at the same time the robot is shrunk to a point. This way a safe distance to the obstacles can be maintained and yet no time is wasted by driving suboptimal paths.

The primary goal of path-planning of "staying away from obstacles" translates into "not crossing any line" with this approach, allowing for the special case of appearingly moving the

[39] See [Hart, Nilsson and Raphael 1968].

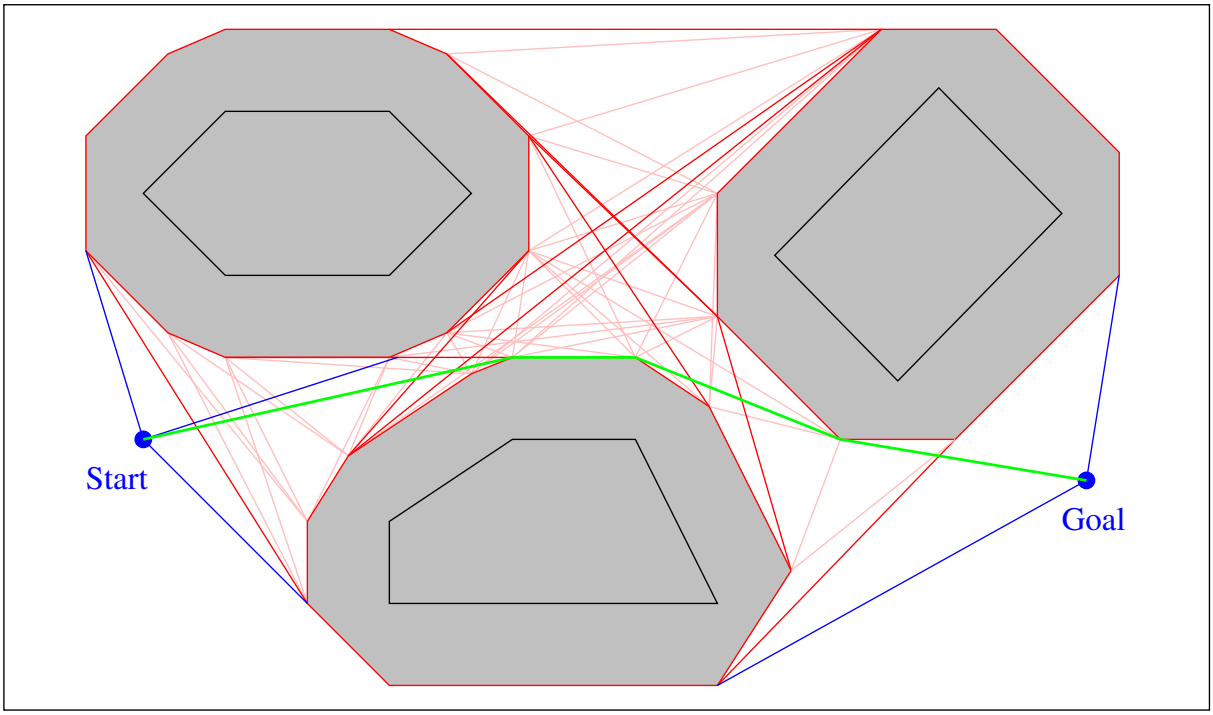


Figure 3.37: Searching a path is done by the A^* algorithm which traverses the graph and finds the optimal (green) combination of path segments.

robot “inside” an object that is given as a polygon. This has to be seen as an advantage because it for example allows smaller robots to be moved on tables using the same map as a larger robot that moves around that very table.

The map is considered to be given because the idea of building it by exploration known as *simultaneous localisation and map building* (SLAM)[40] does not comply with the requirement of having a high precision and safe operation. This is because SLAM requires a learning phase of unpredictable length, It will end up with the capability to go back to positions where it has been before, but it still does not know how to associate these positions with laboratory devices. Searching the devices using the arm and its vision system is for security reasons not feasible, and so a human operator will have to teach the positions of the devices as with an ordinary map.

With the intended field of application of a single laboratory no performance issues arise. The complete map and graphs can be held in computer memory with no problems, even though they are only sparsely filled. If the map should become too large the computational effort can be kept reasonable by splitting the map.

Dynamic obstacles are not handled at this level, but by the collision avoidance in the motion execution level. Possible enhancements might include a separate thread that monitors the visible obstacles and enters them into the map if they are found to be persisting or removes them if they are found to have gone again. This, to distinguish them from really static obstacles, would lead to a layered map.

[40] For example see P. Newman’s work [Newman 1999].

3.4 Motion Execution

Once a path for a motion to a target is computed the next step is to drive that motion, or – in other words – execute a sequence of *motion primitives*. In general, it can not be guaranteed that this is actually the last step because unforeseen obstacles or circumstances may make it necessary to stop half-way and replan parts of the path, however, under the limitations of this work it can. The strategies presented in this section therefore cover the *trajectory generator* (TGEN) executing a functionally complete set of motion primitives including a low-level *collision avoidance*, but not high-level replanning. The motion primitives implement the idea of controlling the platform with a non-linear filter based on velocity profiles, which – contrary to the original Neobotix software – do have a strongly deterministic notion of when a motion is finished.

3.4.1 Kinematics and Motion Types

Depending on the kinematics of the mobile platform and the type of the computed path a lot of different types of motions are possible. In general a mobile platform with a *differential kinematic* as the Neobotix MP-L655 is capable of driving "only" one type of motion, namely

- rotations about any point on the axis going through the platform's drive motors.

Whereas this generic motion type may at first sight appear to be both not very powerful and yet complex to implement it includes two interesting special cases. Since these special cases are much simpler than the generic case and yet exactly the ones which are needed to build a functionally complete set of motion primitives the list needs to be rewritten to include them. Updated, the MP-L655 is therefore capable of driving

- circles with a radius r,
- rotations on the spot (a circle with radius 0) and
- translations straight ahead (a circle with radius ∞).

This set of motion types is already functionally complete[41], and so the last motion type to be mentioned is not explicitly required, but only a nice-to-have. Adding to the above, the MP-L655 is also capable of driving

- splines (or other curves)

because they can be locally approximated by a circle. Note again that with a differential kinematic it is not possible to move the platform sideways without rotating first – such a motion can not be expressed by a rotation about a point on the vertical axis going through the platform centre.

Before presenting the velocity profile based filter used in this work it shall first be shown how the original vendor software implemented motion execution.

[41] It can be trivially seen and has been shown in section 3.3 that a point-sized object can reach any target with only rotations on the spot and translations, whereas arbitrary shaped objects may require rotations about arbitrary points.

3.4.1.1 Neobotix Software

As has been mentioned, the original GENCONTROL software does not use a path planner and can therefore not at all reach targets which are more than just trivially obstructed. Instead it uses a three-staged motion execution with an integrated obstacle avoidance to move around those obstacles it can deal with. The three stages of motion execution are:

1. Rotate towards the target until the difference in orientation is below some threshold,
2. drive towards it until the euclidian distance is below some threshold and
3. rotate in target orientation until the difference in orientation is below some threshold.

These stages are not fixedly executed one after another, but newly selected in each cycle according to the conditions – see figure 3.38. This is because it is possible that in stage 3 the euclidian distance condition is violated again due to slippage or other kinematic problems. In this case the TGEN switches back to stage 2, trying to get back to the target, or – more likely – even to stage 1 to orientate the platform towards the target again. Only if all three conditions are satisfied the TGEN stops sending commands to the drive motors. Since the platform is basically a mass with inertia this does not necessarily mean that it immediately comes to a rest, and if during the final deceleration one of the constraints is violated again the procedure is restarted right away.

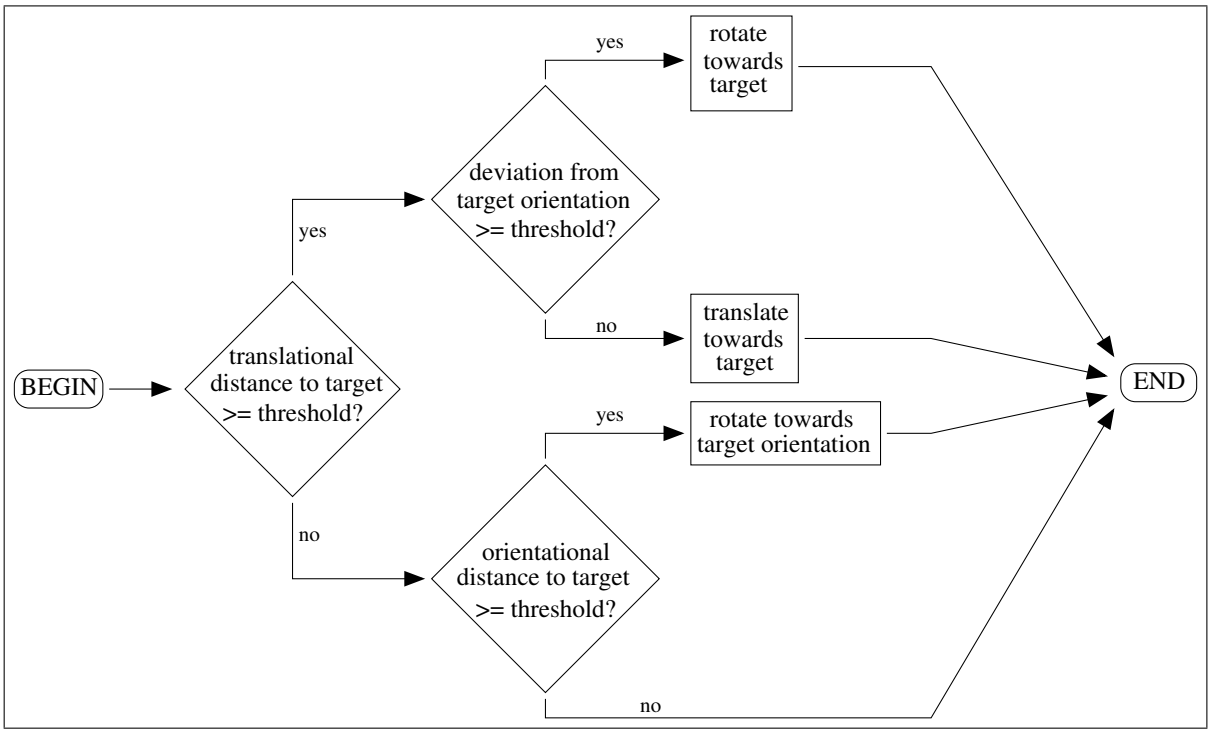


Figure 3.38: Tracking a target in the original GENCONTROL software. BEGIN and END mark a single cycle of the control software, not the entire motion.

This means that GENCONTROL *tracks* a target rather than execute a deterministic motion towards it – it does not have a true notion of a "motion". As such, termination of the "motion" can only be determined by additionally observing the above constraints at a higher level, plus

ensuring that the drive wheels are actually no longer moving. This approach is of course very error-prone, giving one of the reasons for the decision to replace GENCONTROL.

Another reason is that the final termination of the TGEN loop is of course not guaranteed by this approach. In fact the platform has been observed to get trapped in endless motion if the catch radius around the target is smaller than the accuracy of the localisation. Such a behaviour is of course untolerable in the given setup.

The newer Neobotix software "PLATFORMCONTROL" which is now shipped with the MP-L655 does – though being a complete rewrite – does not correct all these issues. Instead of using the three-stage-approach for linear motions it can drive the robot on a spline curve, but since it – at the time of writing – still has no motion planner the spline has to be externally given. Also, it still has the problem of getting endlessly trapped in non-trivial situations. It is therefore also not suitable for the given requirements.

In addition to that, both the GENCONTROL and PLATFORMCONTROL run under the Windows operating system which, since the robot arm control runs under the Linux operating system, means that an additional PC has to be installed on the mobile platform and additional communication has to be implemented. Since this unnecessarily complicates the setup it presents a final reason to implement a replacement.

3.4.1.2 Tracking & Inverse Kinematics

Considering alternative implementations, there is at least one other important thought to consider: Moving means to change the position, and one way to do a motion is to precompute a trajectory, i.e. the position as a function of time. Motion execution then means to time-step this function and apply inverse kinematics to yield a set of commands to feed to the drive mechanism. This is the way it is usually done for robot arms where there is a strict kinematic coupling (i.e. no slippage) between the drive mechanism and the space in which the arm is to move. A mobile platform, however, does not have this strict kinematic coupling, and therefore this approach can not be applied directly.

One typical suggestion to overcome this problem is that only a virtual point is moved and the platform is requested to track that point by means of some additional controller. Depending on the controller, this means that even though the virtual point is moving with constant steps the platform itself will not – if slippage occurs. Where the maximum velocity is likely to be limited, meaning that the platform will not catch up if it lags behind, reduction of the velocity will surely occur if the platform overshoots.

As a result this approach does work in terms of moving the platform along the trajectory, but may do so using a non-constant velocity and therefore leads to jerky motions. A more sophisticated approach will of course avoid this problem in the first place.

3.4.1.3 Conclusions

Because of these – partly fatal – disadvantages of the commercial software a different approach had to be implemented for this work. It was especially designed to be

- compatible with rotations and translations,
- have a deterministic concept of the termination of a motion and in particular

- prevent endless loops. It should
- not suffer from the tracking problem,
- have basic collision avoidance, but
- need not have obstacle avoidance capabilities.

The central means to achieve these goals is a trajectory generation filter that uses velocity profiles.

3.4.2 Trajectory Generation Filter with Velocity Profiles

Both rotations on the spot and translations can be seen as travelling a distance in one dimension – the angle of orientation in case of rotations and the projection of the position on the desired trajectory in case of translations. As is shown by Lloyd and Scherer[42] this allows for a simple velocity profile to be computed for that travel.

3.4.2.1 Velocity Profiles

The physical foundation behind velocity profiles is that the distance d travelled is the integral under the velocity curve $v(t)$.

$$d(t) = \int_0^t v(t)dt$$

This alone is not a constructive rule as it does not specify how the curve should look like. In order to construct a proper curve additional information is needed, which does exist in the requirement to get to the target as fast as possible.

In case of static profiles (i.e. those with zero initial velocity) this means to ramp up (accelerate) to some peak velocity and then to ramp down (decelerate) to stop exactly at the target. As can be seen in figure 3.39(a) such a profile makes the computation of its integral quite trivial. The distance d travelled under such a profile is given as

$$d = \frac{v_p^2}{a}$$

and the time t_s needed for the travel as

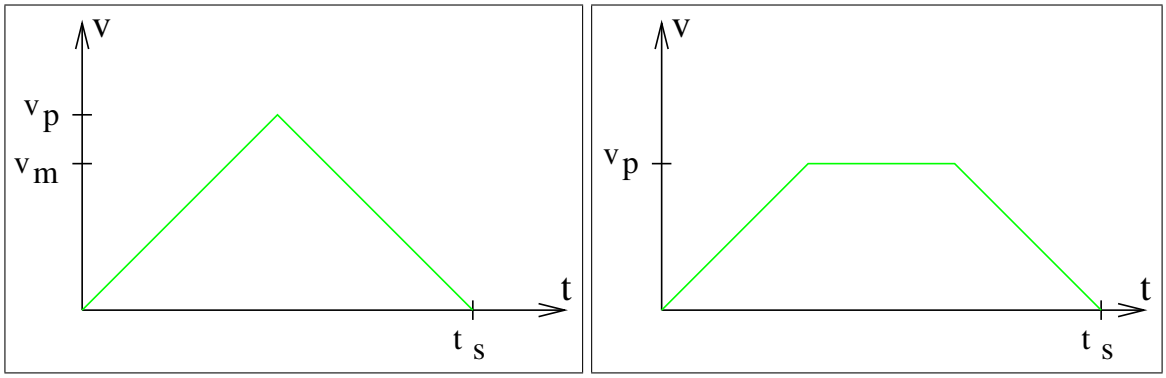
$$t_s = \frac{2v_p}{a}$$

The peak velocity v_p can easily be computed from the above as

$$v_p = \sqrt{a \cdot s}$$

In most cases this v_p will exceed the maximum allowed velocity of the platform, so it has to be clipped. The resulting profile has a phase of constant velocity as in figure 3.39(b), making the integral computations slightly more complex because the system can be in any of three different states:

[42] See [Lloyd 1998] and [Scherer 1998], chapter 3, p.14 ff.



(a) non-clipped (b) clipped

Figure 3.39: Simple static profiles.

1. The system is accelerating to the maximum velocity.
2. The system is travelling with constant maximum velocity.
3. The system is decelerating to stop at the target.

The profile computations are the same for all these states. The distance d travelled under the profile is given as

$$d = v_p t_s - \frac{v_p^2}{a}$$

and – since v_p is known to be the maximum allowed velocity – the time t_s needed for the travel as

$$t_s = \frac{d}{v_p} + \frac{v_p}{a}$$

From these v_p can be computed as

$$v_p = \frac{at_s - \sqrt{a^2 t_s^2 - 4ad}}{2}$$

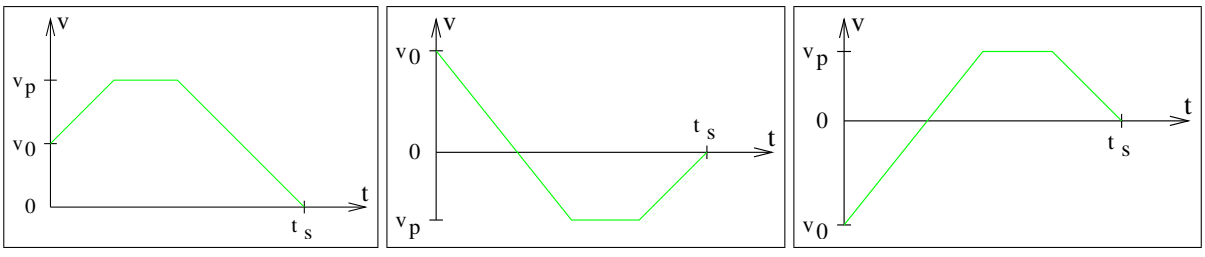


Figure 3.40: Three possible different types of dynamical profiles.

As has been said, these static profiles are for zero initial velocity only. If applied, they will put a system in motion and so this condition will no longer hold in the next cycle. What is needed as a generalisation are dynamic profiles, i.e. profiles with a non-zero initial velocity. This dynamic case is a bit more difficult as there are in fact three different types of profiles possible, as can be seen in figure 3.40:

1. $v_p \geq v_0 > 0$: The system is going towards the target in any of the the above three stages. It will come to stop at the target.

2. $v_0 > 0$, $v_p < 0$: The system is going towards the target, but the initial velocity is too high and it will overshoot if it must not violate the maximum allowed acceleration. It will have to go backwards to finally stop at the target.

3. $v_0 < 0$, $v_p > 0$: The system is going away from the target and has to decelerate to turn around into a motion towards the target.

The types 2 and 3 actually do not occur with the mobile platform control because of reasons that will be shown below, and so only the type 1 will be further analysed here. This type can without loss of generality be divided into a ramp-down from the initial velocity v_0 covering the distance $d_0 = \frac{v_0^2}{2a}$ plus some extra distance $d^* \geq 0$. The fastest way to travel this extra distance is again to accelerate to some peak velocity v_p, yielding a profile as in figure 3.41.

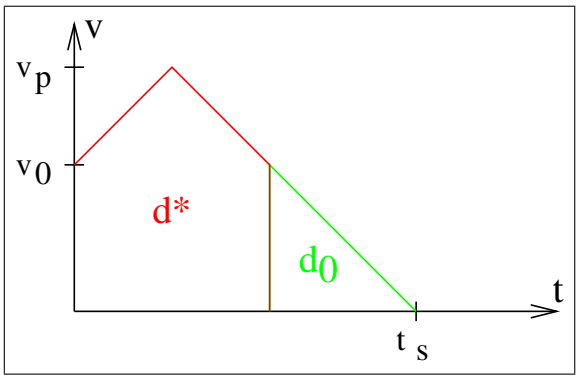


Figure 3.41: The extra distance d^*.

The peak velocity v_p of this profile can be computed as

$$v_p = \sqrt{ad + \frac{v_0^2}{2}}$$

as long as $d \geq \frac{v_0^2}{2a}$ holds, because if $d < \frac{v_0^2}{2a}$ we are not in a type 1 profile at all. If $d = \frac{v_0^2}{2a}$ we are in the special case of $d^* = 0$, for which $v_p = v_0$ holds – a case that consist **only** of a ramp-down.

Again, this v_p has to be clipped against the maximum allowed velocity. Depending on whether the clipped v_p is larger or smaller than the initial v_0 two major cases have to be distinguished:

1. $v_0 < v_p$: The profile first ramps up to v_p, optionally stays at that velocity for some time and then ramps down to stop at the target (see figure 3.42 left). The distance travelled can be computed as

$$d = v_p t_s - \frac{v_p^2}{2a} - \frac{(v_p - v_0)^2}{2a}$$

The only parameter that is still missing is the time t_s needed for travelling the motion, which can be computed from the distance as

$$t_s = \frac{d}{v_p} + \frac{v_p - v_0}{a} + \frac{v_0^2}{2av_p}$$

2. $v_0 > v_p$: The profile directly ramps down to v_p, optionally stays at that velocity for some time and them ramps down again to stop at the target (see figure 3.42 right). The distance travelled can be computed as

$$d = v_p t_s - \frac{v_p^2}{2a} + \frac{(v_0 - v_p)^2}{2a}$$

and the motion time t_s as

$$t_s = \frac{d}{v_p} - \frac{v_0^2}{2av_p} + \frac{v_0}{a}$$

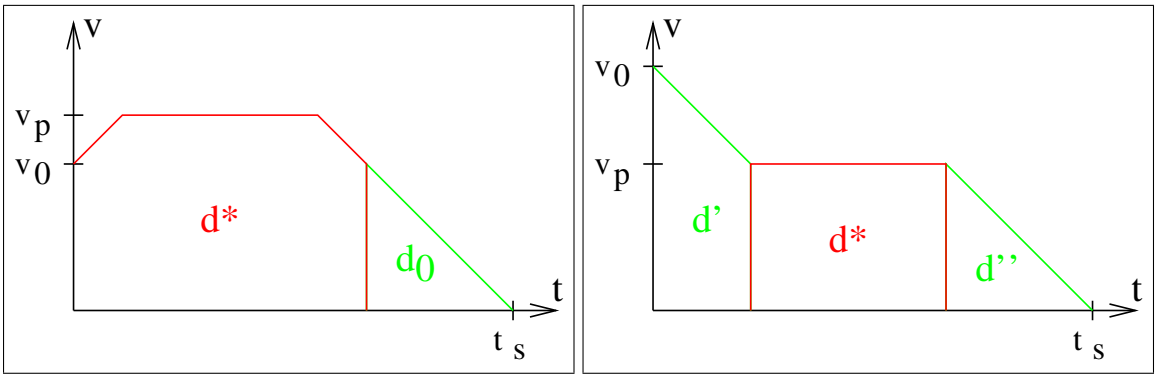


Figure 3.42: Two different cases of covering the extra distance d^*.

These parameters are all that is needed to construct a one-dimensional profile. Since [Scherer 1998] originally used these profiles to control robot arm motions he also defines stretching a profile, that is enlarging its t_s to synchronise several dimensions of a multi-dimensional profile, which involves possible switches between the above two types. However, stretching is not needed for one-dimensional profiles, therefore its description has been dropped here.

Also only briefly mentioned is that while a one-dimensional profile can be proven to be stable, a multi dimensional in general can not. [Scherer 1998] uses multi-dimensional profiles to move a point in 3d-space to a target, but under certain restrictions. If used without restrictions a multi-dimensional is likely to result in an *attractor* system and converge to a "stable" *orbit* around the target, rather than a point.

3.4.2.2 Position Setpoints

Depending on the desired output the application and evaluation of these profiles differs. Lloyd and Scherer use them to generate commands for a PUMA robot arm which accepts *position setpoints*, so in their application they really integrate the profile for some cycle time t_c. This increment in position is then added to the current position to obtain a new setpoint.

A motion is terminated once the profile is used beyond its length t_s. Since after termination of the profile the target position value is continuously output to be tracked by the motor controllers it does not matter if they actually reached the target at t_s or not. The typical PUMA arm motor controller will in fact lag a bit behind its desired target value, but this effect can and is usually ignored as it only makes up for fractions of degrees.

This usage of velocity profiles yields no problems except that the target velocity that is also provided by the profile is ignored because the PUMA can make no use of it. This also means that this value is lost in the next cycle when a new profile is to be computed. Instead, they use the differential position as reported by the robot as v_0 for the new profile. This v_0 will be smaller than the actual velocity in acceleration phases and larger than it in deceleration phases. However, Scherer shows that this does not make the filter unstable[43].

3.4.2.3 Velocity Setpoints

Applied to the MP-L655 mobile platform the differential position as obtained by integrating the profile can not be used as a control command because the motor controllers do not support position setpoints, but only *velocity setpoints*. Instead of implementing an additional PID controller around each motor that allows feeding of position commands this task is laid upon the profile itself. This means that it is no longer the finally desired target entity that is controlled, but its first derivative, and so problems are to be expected.

Note that – contrary to a robot arm – with a mobile platform there is slippage between the motors and the entity that is to be controlled. This means that the profile really has to be newly computed in each cycle because no assumption about the remaining distance to be travelled can be made. This raises the question of which velocity to use as initial velocity v_0 of the profile. Using the actual velocity as obtained by differentiation of the position triggers a problem at the start and end of the ramps:

- At the end of the deceleration ramp, assume that the last profile cycle exactly stops at the target with velocity zero. The output to the motors will be a zero velocity, meaning that in theory the platform will not move that last remaining distance. In practice it will still move a little bit because the platform's inertia will forbid it to stop immediately. In such a case the motion is terminated as soon as the profile is used beyond its length t_s and a few cycles are spent waiting until the platform really comes to a stop. The remaining error is typically in the range of very few degrees of wheel position and can well be tolerated.

- At the begin of the acceleration ramp the situation is a lot more severe. In the very first cycle the profile will yield a quite small target velocity. If this velocity is too small to overcome the mass inertia the platform will not move. This, however, means that in the next cycle the actual velocity as the differential velocity between two position will be zero, because the position has not changed. As a result, in the next cycles the situation will always stay the same (see figure 3.43): The position has not changed and the velocity is zero – the platform will *never* move.

This problem is inherent with discretised systems, in particular when only the first derivative of the desired entity is controlled. In order to overcome it, the actual velocity must not be taken to compute the profile. Instead, the target velocity from the last cycle is used as initial velocity for a new cycle. As can be seen in figure 3.44 this means that the initial v_0 will get larger in every cycle and eventually overcome the mass inertia.

A visualisation of a control cycle of this approach can be seen in figure 3.45. Since the profile is always recomputed with the actual distance to be travelled this approach works without causing larger positioning errors. It does, however, add to the final positioning error, leading to

[43] See [Scherer 1998], p.27.

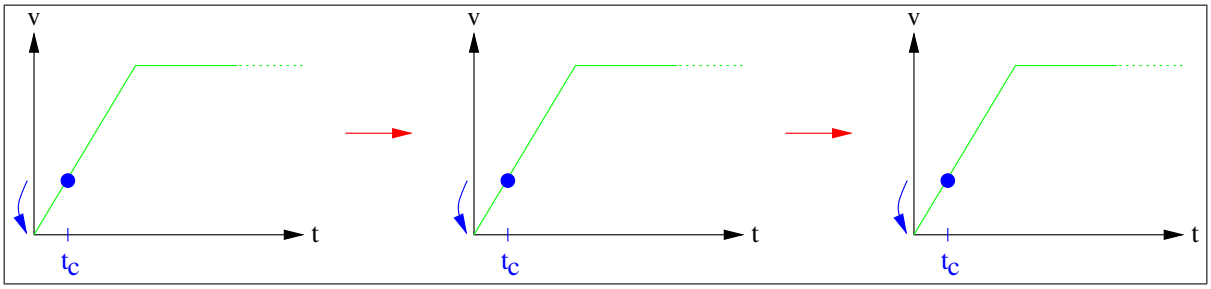


Figure 3.43: Using the real velocity as initial velocity for a profile may let it get stuck.

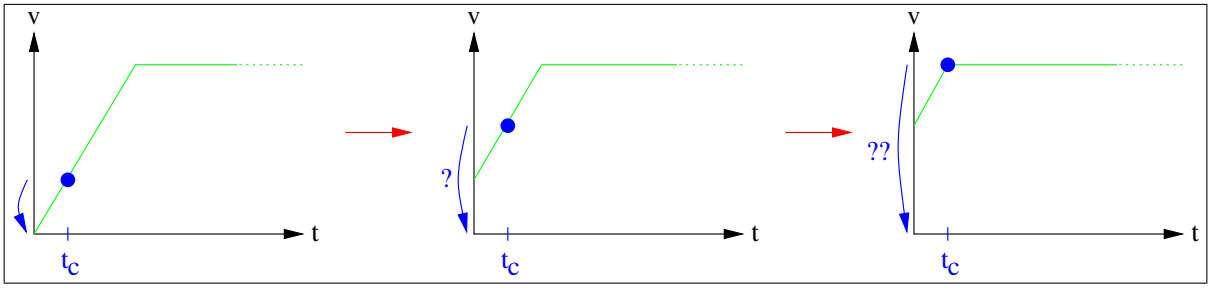


Figure 3.44: Using the target velocity from the last cycle as initial velocity for a new cycle overcomes the inertia and ensures that the platform starts to move.

that the termination conditions have to be modified. Therefore, a motion is forced to terminate when either

- the current position is beyond the target,
- the output velocity is zero (which may happen a bit short off the target, but not because of an active collision avoidance) or
- according to the timeline the target should be reached in this step (regardless of whether this really is going happen or not).

It is because of this focus on terminating a motion once it has come close enough to or passed a target without trying to go back to it that the above mentioned type 2 and 3 profiles actually do not occur in the profiles as used in this work. As a result of this the real position may differ from the desired target when the profile has stopped.

3.4.2.4 Summary

It has been shown that velocity profiles can be used to implement a TGEN as a filter. In case of one-dimensional problems like during rotating or translating a mobile platform it is theoretically stable, whereas in the generic multi-dimensional case it needs additional constraints. The practical stability problems that arise from the fact that the initial ramp-up may get stuck due to discretisation or mass inertia can be overcome with only slight modifications.

The resulting implementation puts a strong emphasis on deterministic termination of a motion, even if this means to slightly deviate off the target. It therefore needs no catch radius and no knowledge about the overall accuracy to determine motion termination and can by design not get trapped in an endless loop like the original vendor software.

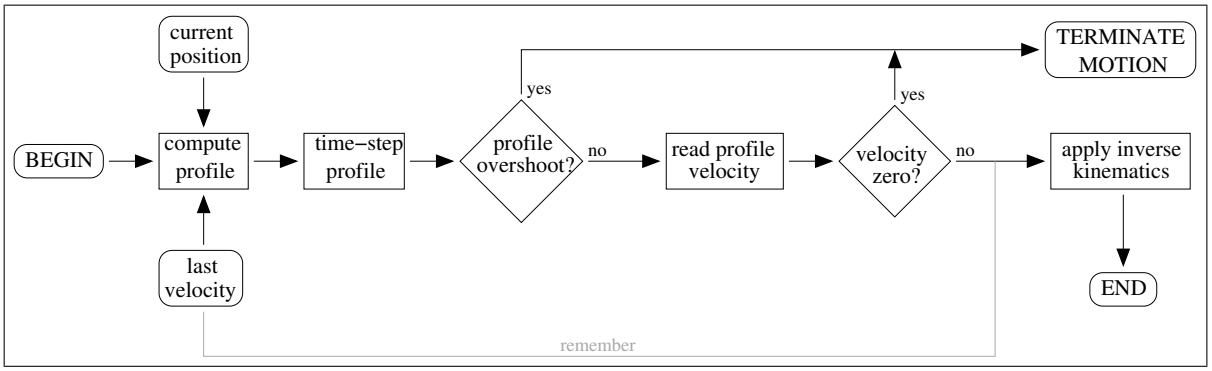


Figure 3.45: Moving to a target with the new approach presented in this work. Again, as in figure 3.38, BEGIN and END mark a single cycle of the control software, not the entire motion.

3.4.3 Absolute Rotations

Absolute rotations – i.e. rotations to an absolute orientation – are needed to change the orientation of the platform between path segments or at the end of a path. They are implemented by a one-dimensional profile for the orientation angle. During each cycle the current orientation is obtained from the localisation, the profile recomputed and the target velocity send to the motor controllers. The rotation profile is terminated according to the above conditions.

The rotations are silently assumed to be "on the spot", though this is neither checked nor enforced. Due to slippage or other kinematic problems it happens that during the rotation the *position* changes slightly. As can be seen in figure 3.46, this translational deviation depends on the absolute rotation angle. It is at most 2 cm for a rotation of 180 °, so for rotations between path segments it can be ignored.

For rotations to go into target orientation at the end of a path such a deviation can not be tolerated. Because of this, the approach to a device is implemented in a way that avoids rotations as the last motions. Instead of directly going to a target the platform first goes to a point some *approach distance* in front of the target and then goes the last part in a straight line. This way at the target at most only a very small rotation is necessary to compensate for errors that have occurred during the final approach.

Currently only rotations to a target *angle* are implemented, not to a target *point*. The rotations between path segments are in fact rotations towards a point (the end point of the next segment), but instead the pre-computed target angle is used. This does make a difference because during rotation the position of the platform may shift and therefore the target angle too. Due to the lacking target point rotation mode this effect is currently not compensated for and so the setup has to be arranged in a way that it can be ensured that rotation to not take place above sinks, drains or other bumps in the floor.

3.4.4 Absolute Translations

Absolute translations – i.e. translations to an absolute point – are needed to drive the platform along a path segment from a start point $\vec{p}_0$ to an end point $\vec{p}_e$. Although this is a motion in a 2d-world it basically is only a 1d-problem because the robot is desired to drive the path segment as a straight line. This requires that the platform is basically oriented towards the target – if it is not, a rotation has to be issued first. Once this has been assured the translation can start.

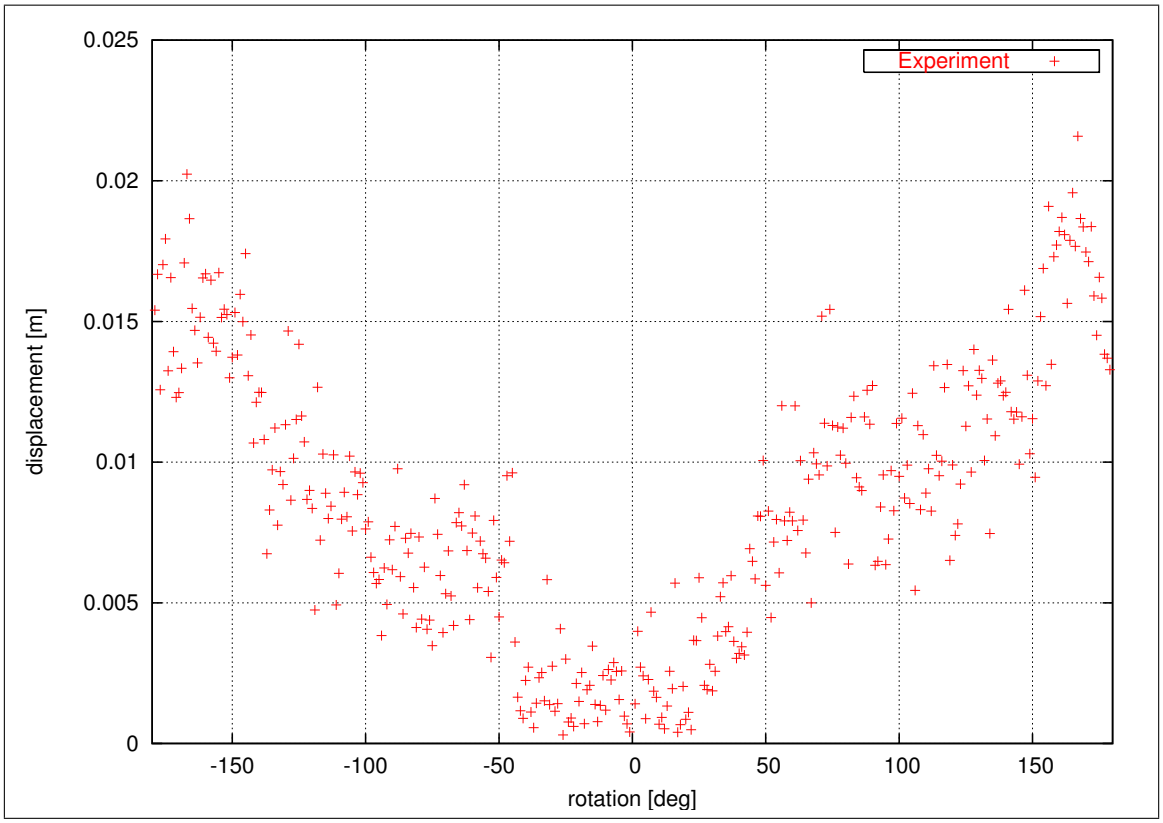


Figure 3.46: Translational displacements during rotations.

First, the length l_t of the desired trajectory $\vec{p}_t$ and the trajectory itself are computed as

$$\begin{aligned} l_t &= \|\vec{p}_e - \vec{p}_0\| \\ \vec{p}_t &= \frac{\vec{p}_e - \vec{p}_0}{l_t} \end{aligned}$$

Then, in each cycle t the current position $\vec{p}(t)$ and velocity $\vec{v}(t)$ of the platform are projected onto the trajectory as

$$\begin{aligned} p_t(t) &= \vec{p}(t) \cdot \vec{p}_t \\ v_t(t) &= \vec{v}(t) \cdot \vec{p}_t \end{aligned}$$

A 1d-profile can then be computed to move the platform along the trajectory and bring it to rest after some time t_s so that

$$\begin{aligned} p_t(t_s) &= l_t \\ v_t(t_s) &= 0 \end{aligned}$$

holds.

During the travel it may happen that non-zero position and/or velocity components perpendicular to the trajectory occur because of slippage and/or other kinematic problems or inaccuracies. In order to compensate for these a PID controller is added ontop of the profile to try and bring these components back to zero, i.e. the platform back to the trajectory. This controller is only parameterised to perform small corrective motions and not full rotations, so the platform really has to be basically oriented towards the target.

Motions are terminated according to the above conditions, which – due to the PID controller producing commands perpendicular to the trajectory – means that not only a small deviation along the trajectory from the target may occur, but also a small deviation off the trajectory. Basically, the platform can be expected to arrive near the target with what can be approximated by a gaussian distribution. The positioning error is typically within 1 cm, with worst cases of up to 5 cm. This error must not be mixed up with the localisation error, which is typically less than 1 cm, but has to be added to the above to yield the absolute accuracy. The overall performance of the platform control is shown in more detail in section 3.6.

It is mainly because the PID controller forbids arbitrary deviations off the desired trajectory that the final positioning error is so low. The original GENCONTROL software did not have the notion of a desired trajectory and therefore allowed for arbitrary deviations off it. This means that the platform may arrive at the target with an arbitrary orientation, and since rotations to correct this orientation in turns result in changes of the position it suffers from a much larger positioning error and the potential endless-loop-problem.

With the approach shown here these problems do not have to be compensated because they do not arise in the first place. Also, no catch radius and no knowledge about the accuracy of the localisation are needed to ensure a deterministic behaviour – the only effect a bad localisation would have is to make the accuracy of the positioning bad too.

3.4.5 Relative Translations

Relative translations – i.e. translations by a distance – are only needed to approach to and depart from the *automatic charging station* (ACS). They are based on absolute translations with three differences. They

- disable the collision avoidance,
- disable the PID controller and
- do not use the real position as feedback.

The reason for disabling the collision avoidance is that the ACS is mounted to a pillar in the laboratory and approaching it means that the normal safety distance has to be violated. As a consequence relative translations should not be used in situations where unexpected obstacles (humans) may be present.

The reason for disabling the PID controller is that due to the design of the ACS it is not favourable to approach it with an orientation that is not exactly perpendicular to it. In case the orientation is only a little bit sideways the electric contacts in the plug may not properly connect with the socket. Therefore, since during short translations only very small orientation errors are likely accumulate, it is favourable to use a short approach motion from a point directly in front of the ACS and tolerate the small orientation error rather than to try and compensate it, which may temporarily lead to even larger orientational deviations. As a consequence the ACS has to be mounted at a place where such an approach in possible, which in particular means that the floor near it must be free of sinks, bumps or other troublesome points.

The reason for not using the real position as feedback is that the “real” distance to drive can not be given. The position of the approach point and the ACS are of course known, but if a motion to the precise target is issued one of the following two problems may occur:

1. If the distance was too short no sufficient electrical contact can be made. Since the ACS is not equipped with an ampèremetre indicating that current is actually flowing such a situation can not be properly detected. The only thing that could be detected is whether the battery voltage is increasing or not, but since this has to be measured over a longer period of time to be absolutely sure[44] it is a very cumbersome measurement. This is especially true if the battery is already almost full and the change in voltage during loading is therefore very small.

2. If the distance was too large the platform is stopped by "bumping" into the ACS before reaching the target. Since the profile would see that it has not yet reached the target it would therefore continue producing motion commands. This means that the motion would run indefinitely with the platform standing in front of the ACS with spinning wheels.

Instead, if the true position is not taken as feedback for the profile the motion can be set up to go one or two centimetres "into" the ACS. The profile will not see that the platform has bumped into an obstacle and will continue to drive towards the target until it stops. As a result the platform can be assured to have established proper electrical contact against the spring mechanism used in the ACS, and the fact that the wheels spin for a second or two does not hurt – this also happens at very bumpy floors.

3.4.6 Collision Avoidance

Collision avoidance is actually not implemented in these motion classes, but one level higher in the TGEN that calls these classes. The TGEN computes a simple *velocity scaling factor* based on the distance d to the nearest obstacle

$$f(d) \quad = \quad \tanh(c \cdot (d - d_{\min}))$$

which is clipped to not become negative[45]. The scaling factor $f(d)$ is then applied to the velocity in each cycle. Some examples of this factor for different parameters can be seen in figure 3.47.

If the platform was slowed down in order to avoid a collision the profile has to be notified of this because it has to use the correct velocity in the next cycle. If it would not know about the slowdown it might see that the distance has not changed, but would assume to still be at maximum velocity. This means that as soon as the obstacles goes away it would try to resume the motions with full speed instead of ramping up, which is not a good idea.

Also, the profile has to know about the slowdown in order to not erroneously believe that the motion termination condition checking for whether the output velocity is zero has become true. During a slowdown caused by the collision avoidance this one condition is therefore explicitly disabled. Since the other two conditions only become true if the target is explicitly crossed this

[44] If, for example, the platform has recently been moving with full speed (a lot of current has been drawn from the battery) and the battery has already been quite low the voltage will increase as soon as the platform (the current withdrawal) is stopped. This is because of the way lead-acid batteries work: Their chemical agent represents an electric resistor that is increased when the battery is discharged. Since a withdrawal of current causes a loss of voltage over that resistor the available external voltage gets lower. If the withdrawal of current is reduced the external voltage may therefore rise again. The absolute voltage of course stays the same.

[45] Care has to be taken that the parameters c and $d_{\min}$ do not lead to too much deceleration at the nominal safety distance as kept by the path planner, because otherwise the platform will mistake the objects the pathplanner is moving around as obstacles and slow down.

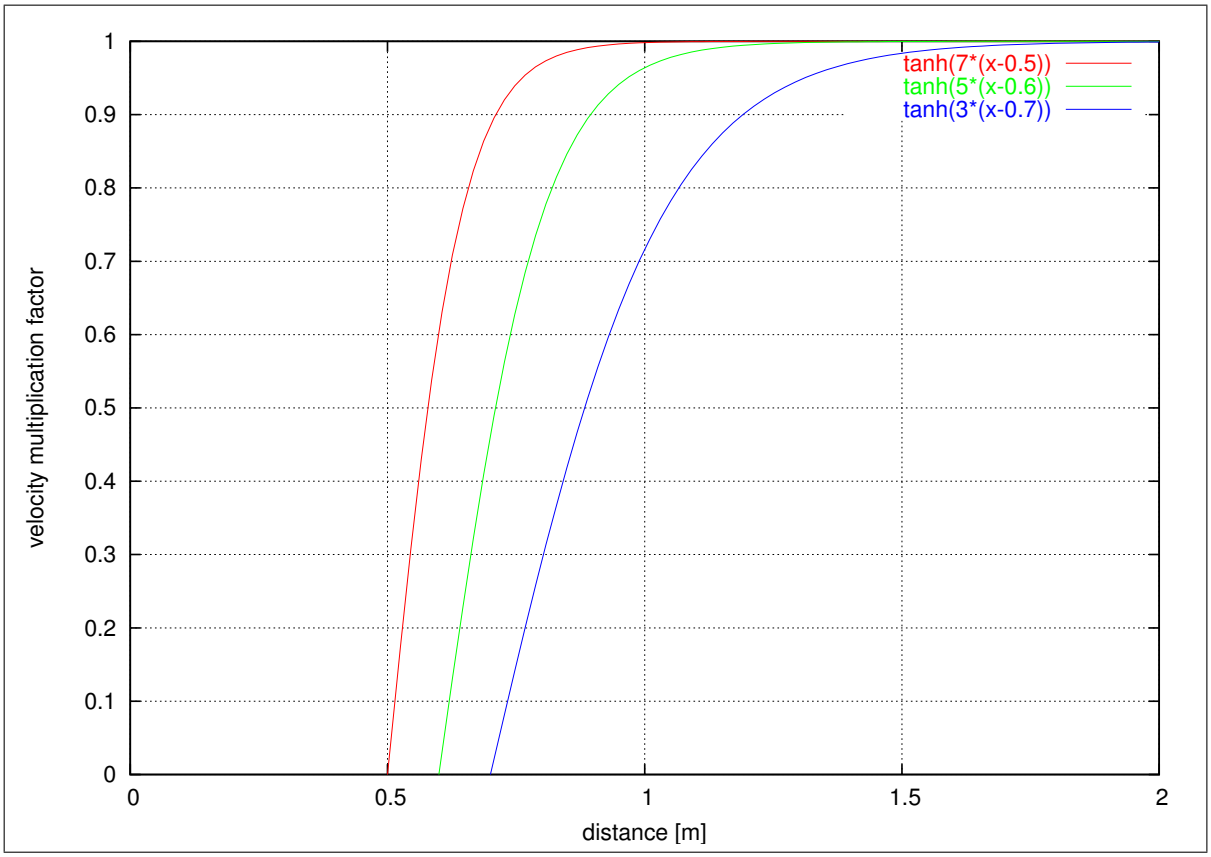


Figure 3.47: Velocity scaling factors used for collision avoidance.

means that under worst circumstances it is possible to pull the platform slightly more beyond the target if a moving obstacle is involved than it would have gone with no obstacles nearby. Since this is a consequence of a required safety mechanism it can not be called a disadvantage.

Currently, no models about objects and their motion are derived from the scans and therefore the collision avoidance does not make a difference between objects that are moving towards the platform, standing still or moving away from it. As a result the parameters have to be chosen conservatively to avoid collisions with humans moving towards the platform. Since the presence of humans in the laboratory is considered to be the exception this is also not considered a problem.

3.4.7 Realization & Enhancements

The rotations and translations are implemented in C++ classes encapsulating the profile. They are both derived from the generic class `_CGENMOTION` declared in the header file `genMotion.h` as in table 3.3.

The basic working principle is that the outer class (`CGENBASE`) sets the `_pos` and `_vel` entries to their current values. Then it calls the virtual `Step()` method. It is up to that method to implement everything that is needed, including computing profiles. The parameters required for this can be stored in the motion class (`CROTATION` for rotations and `CTRANSLATION` for translations).

More actually, these classes are not called directly, but through a special class, `CMOVE`, which encapsulates some typical combinations of rotations and translations as they are needed for

```
class _CGENMOTION
{
  friend class CGENBASE;
protected:
  CVEC _pos;
  CVEC _vel;
  double _transVel, _rotVel;
public:
  virtual bool Step (void) = 0x0;
  void GetVelocities (double &transVel, double &rotVel)
  {
    transVel = _transVel;
    rotVel = _rotVel;
  }
};
```

Table 3.3: Excerpts from the generic motion C++ class `_CGENMOTION`.

moving a path. The `Step()` method of `CMOVE` is then responsible for updating `_pos` and `_vel` of the current second level motion class and calling their `Step()` method. The motivation for this encapsulation is that it makes a complete path with all segments look like one single motion only to the calling `CGENBASE`.

Using this interface additional types of motions can be very easily added, simply by writing a new motion class derived from `_CGENMOTION` and implementing a new `Step()` method. The above mentioned classes are fully functional for the laboratory setup, yet the most prominent enhancements shall be briefly mentioned.

Relative Rotations : One motion type that can be easily added are relative rotations, i.e. rotations by some angle, regardless of what the current orientation is. Relative rotations are helpful if for example the robot is to be moved by interactive remote control to perform monitoring or inspection tasks in a future application. The necessary motion class will almost be a copy of `CROTATION` with only the initial target orientation computation being changed.

Absolute Rotations to Points : Another interesting motion type are absolute rotations to points. As has already been mentioned, the rotations between segments of the path are actually rotations to the end point of the next segment, but are currently only emulated by rotations to the fixed orientation of the next segment's end point as seen from the current segment's end point. This does not take into account that during a rotation the position of the platform may shift due to slippage and/or other kinematic problems. A motion class that does take this into account will also be based on `CROTATION` with the only major difference that the target orientation angle is also newly recomputed in each cycle. This will not make the filter unstable if the two points are not too close together (or – worse – identical), in which case care has to be taken.

Splines : Finally, the most complex motion type are spline motions, which are interesting for driving smooth paths. As has – again – already been mentioned, splines can be locally approximated by circles, and therefore the platform can drive spline motions and the

necessary motion class can be added to the control software. There are more and more complex parameters required for defining a spline, but since each motion class can declare its own parameters this is not a problem. Using a spline motion class is, however, not that straightforward as using the other classes. The most likeliest implementation will probably move a virtual point along the spline and then let the platform track that point, which introduces another layer of indirection. Also, it will require a different path planner because the spline curve has to be ensured to keep a safe distance from all obstacles. Using the existing path planner and "just" smoothing the corners of the path will violate this safety distance.

3.4.8 Summary

It has been shown that the final step in moving a mobile platform, the actual motion execution, is a 1d-problem if only rotations on the spot and translations are involved. It has also been shown that velocity profiles are a suitable means of executing such motions, as opposed to tracking a virtual target using explicit inverse kinematics. They avoid jerky motions because they always drive the platform with the maximum velocity (outside acceleration and deceleration phases), regardless of how much distance is actually covered in each cycle. It is because of this that they can – contrary to tracking approaches – be guaranteed to be time-optimal.

The resulting implementations of rotations and translations put a strong emphasis on deterministic termination of motions, even if this means to slightly deviate off the target. Small deviations can be far better tolerated than getting trapped in an endless loop as with the original vendor software.

The C++ interface to motion execution is both powerful enough to implement all the aspects of the presented approach and simple enough to allow for easy addition of new motion types.

3.5 Future Work

In this section a few alternatives to the hard- and software used in this work will be briefly presented, like alternative sensors, localisation methods and path-planning issues. A few words will also be spend on behaviour based control, which has been completely omitted from this work despite being an interesting and active area of research. It will be discussed why these alternatives might or might not be be improvements to the current implementation.

3.5.1 Sensors

The biggest problem that remains is that the localisation is only an estimation process which can not be guaranteed to tell the truth. The current implementation based on triangulation between known points can, as has been shown, actually diverge unboundedly once the association of the laser marks is lost. This is obviously a question of the sensors involved, and so a few alternatives have to be discussed.

Ultrasonic Sensors

One of the most prominent sensors on mobile platforms are ultrasonic sensors. They operate on the same principle a bat uses for navigating: They emit pulses of ultrasonic sound and listen

for echoes. From the delay in time after which the echo arrives they can compute the distance to the object that reflected the echo.

Ultrasonic sensors are cheap, but of limited use because while they can tell the distance to an object they can not tell the direction of the object. Their receptive field is basically a 3d-cone with a more or less wide opening angle. Mounted on a mobile platform they are widely used as *proximity* sensors to avoid collisions in 3d-space, but not for precision localisation purposes.

Infrared Sensors

The other most prominent sensors on mobile platforms are infrared sensors, operating on the principle of detection of reflections of emitted light pulses. Although they can theoretically focus on a single line they are also of limited use because sampling the environment with only a very limited number of measurements does not allow localisation. Also, they somewhat depend on the type and colour of the surface of the object their light falls on[46].

Global Positioning System

GPS alone can not be used as a sole localisation system because, as has been mentioned, its accuracy of as low as 1 m using DGPS techniques is too large for the precise localisation needed in this work. It is, however, an *absolute* measurement of the position and might as such be used to help limiting the error of the localisation which, once the association of the reflector marks is lost, may otherwise become unlimited. If this error can be bounded the basic problem of landmark navigation – that one does not know or looses knowledge about which mark is which – can be avoided.

RF Beacons

Another promising idea for absolute measurements is just being introduced with the recently increasing interest in *radio frequency identification* (RFID) and *transponder* systems. In these systems an active detector emits radio frequency waves that induce a voltage in an otherwise passive transponder. The transponder (a very small device that can be placed virtually anywhere) can use this energy to send back some information – typically only very few bits, but systems with up to several kilobytes are currently in discussion for machine-readable passports. The detector in turns can not only receive this information, but also triangulate the transponder. For example the Götting KG[47] company is building a new system that computes a 2d-position of the transponder. The current drawbacks of this system are that only one transponder can be read at a time and only if it is within a range of up to 10 cm. Still, this could provide a very useful positional information for a mobile platform because at the same time the identification bitstring of the transponder is returned, making it impossible to mistake it with some other transponder as it can happen with the laser landmarks.

[46] This is actually also true for the SICK laser scanners, which can erroneously detect reflection marks at polished metal surfaces or glass as well as fail to detect objects which absorb the laser (the fabric of black jeans has been found troublesome in this aspect). Only by being extremely sensitive in receiving the reflection and using extensive post-processing of the data the sensor achieves its quality – and becomes very expensive.

[47] Fa. Götting KG, Celler Str. 5, D-31275 Lehrte, Germany. `http://www.goetting.de`

Vision Systems

Recently, a lot of attention and research has been spent on vision systems in general, and omnivision system in particular[48]. As has been mentioned these systems usually only yield a very rough estimate of a positional information, and sometimes even only the abstract likelihood of being in one of a limited number of known situations. They are therefore not suitable for localisation directly, but, like GPS, might help in bounding the error of the laser scanner based localisation. The only problem that remains is that, also like GPS, it has to be ensured somehow that the error of these systems is also strictly bounded, which is at least not trivial, if not impossible at all.

Optical Mice

Another interesting principle is that of optical computer mice. Such a mouse uses a minimal vision system to track the texture of the surface it is moved on, and the idea is whether or not this might be used as an odometry sensor for a mobile platform. The only thing that has to be ensured is that the floor does have a sufficient texture, and the only problem that remains is that these mice can probably not be directly used.

Their mechanism currently only works if it is brought very close to the surface in question, and this means such a sensor would have to be attached very close to the floor. One way to do this is to built a mechanism that uses one of these sensors (not necessarily in a mouse case) and presses it to the floor close to the drive wheels[49], but without introducing any possibility of getting stuck behind bumps or other things that may lie on the floor. If this can not be assured, the idea of whether these sensors can be modified or at least their principle used with ordinary cameras to operate at a greater distance to the surface. Anyway, using this principle an odometry could be established that would not suffer from slippage of the drive wheels and therefore be more accurate over long-term measurements. This idea does not appear to have been tried by anyone so far as no literature is known about it.

Separate Odometry Wheels

The last thing to be mentioned here is to use separate wheels for the odometry encoders. If these are mounted on the same axes as the drive wheels but not driven by the motors and allowed to rotate individually they can provide odometry data that does not suffer from slippage if the drive wheels are spinning free. It has of course to be ensured that these wheel are pressed to the floor tightly enough to not allow slippage, but this can be achieved with any means ranging from softer rubber mixtures for the tyres (since they do not have to carry any weight this should not be a problem) to a full separate suspension.

3.5.2 Localisation

Another field of possible enhancements are the algorithms used for localisation, or, in other words, everything that deals with how the sensors are actually used. For example the existing

[48] See [Zhang, Hübner and Knoll 2001].

[49] This is because these sensors can detect translatory movements, but not rotations. If two sensors are placed close to the drive wheel they can be used to support/replace the drive wheel odometry encoders and detect all motions without additional kinematic computations. If they are mounted elsewhere an additional kinematic model will be necessary to translate sensor motion into platform motion. If only one is used and placed in the platform centre no such model would be needed, but then this single sensor cannot detect rotations.

laser scanners can be used for more than just detecting reflector marks. On the other hand alternatives to the EKF might be interesting.

Detection of Walls in Laser Scans

One possible enhancement to using laser scanners is to not only let them detect reflector marks, but walls too. The idea is to detect sequences of scan points that form a straight line and try to match it with the map. Once such a mapping has been established the platform's parallel distance to that line can be computed (see figure 3.48). The position of the platform must be on the parallel, and if more lines are detected more parallels can be built and the position narrowed down using a least square fit (because the parallels are unlikely to intersect in one point).

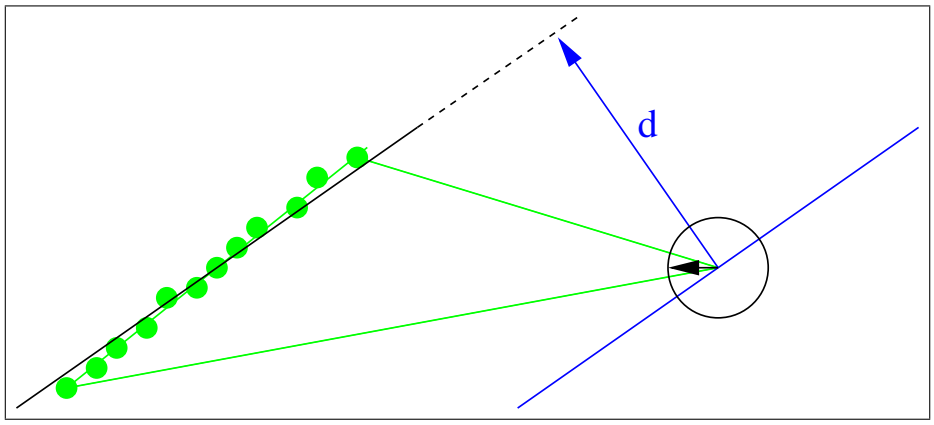


Figure 3.48: Computing the parallel distance to a detected line.

One crucial aspect with this idea is the detection of straight lines in the scan. The individual scan points have to comply with some "straightness" criterion that has to be properly parameterised. The longer the line segments the more reliable the detection will be, but it will not be of any worth if it requires lines longer than any that do exist.

Another aspect is the mapping of these detected lines onto map lines, which also requires a couple of threshold parameters. This, too, is only likely to be free of troubles with long lines. As a result this idea is likely to work in an office-like environment with a lot of hallways (lines), but not necessarily in a laboratory with a lot of reactors, pipes, tables with their legs and/or other things that obstruct the walls or other planar objects.

Unscented Kalman Filter

The *unscented kalman filter* (UKF) is an alternative to the EKF by Julier and Uhlmann[50]. Where the EKF uses a plain gaussian approximation of the probability distribution the UKF uses a set set of sample points to modify that approximation to better fit to the real distribution (see figure 3.49). The UKF therefore is more computationally expensive, but can be shown to be of better performance.

[50] See [Julier and Uhlmann 1997] and [Wan and Merve 2001].

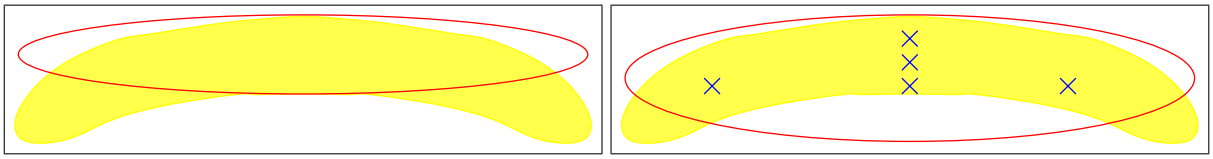


Figure 3.49: Normal EKF approximation of a banana shaped probability distribution (left) and the UKF approximation based on more sample points (right).

Monte Carlo Techniques / Particle Filter

The escalation of the UKF are *monte carlo techniques* like the *particle filter* (PF) by Salmon, Gordon and Smith[51]. The PF uses a set of samples rather than an approximation to represent states and/or distributions (see figure 3.50). Wright, Maskel and Briers say that

> *the particle filter adopts a different approach to Kalman-based filters by sampling a number of hypothesised states for the target; these are the particles. The particle filter does not attempt to model the distribution using an analytic form. Instead, the uncertainty (and so the distribution) is represented using the diversity of the set of particles which simply represent the distribution. Each particle is compared with the measurement and weighted accordingly. Those particles with high weights are propagated and those with low weights discarded.*[52]

A particle in the PF is a system state associated with a weight. Initially all the particles represent the same state and are equally weighted. During a predict step the PF – contrary to the (E)KF – adds gaussian noise to each particle before applying the system model, which leads to a distribution of future states. The idea of the PF is that this distribution can have an arbitrary shape and is thus better suited to model the probability than the gaussian distribution of the (E)KF.

When updating the state with a measurement each of the particles is individually compared to the measurement and its weight updated accordingly. From these weighted particles and the measurement the new state can then be computed, for example by choosing the highest weighted prediction. After the update the sequence is repeated from the start, applying new gaussian noise to each particle and thus slowly spreading them.

If the number of particles should for practical reasons be constant additional manipulations apply: A particle whose weight is below some threshold (is too far away from the optimal solution) is removed and instead the particle with the highest weight is split in two. This way it is also ensured that the particles will not spread into infinity, but remain close to the optimal solution.

During this process no statement can be made about the way the particles organise, in particular not whether they will build clusters or not. If they do, the approach to compute the new state by choosing the highest weighted particle may jump between clusters. This only motivates that in fact a large variety of possible selections or computations can be applied, and the optimal one has to be chosen manually according to the system in question.

As can be seen, the evaluation of a PF is considerably more complex than that of a (E)KF or UKF because a lot of samples are needed, but the gain is even more accuracy in modelling[53].

[51] See [Salmond, Gordon and Smith (1993)].
[52] See [Wright, Maskell and Briers 2003].
[53] See [Doucet, Freitas and Gordon 2001].

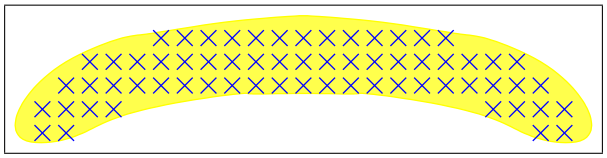


Figure 3.50: Particle filters (PF) use a set of samples to represent any shape of probability distribution. Note that this image uses a grid to visualise the particles, but actual PFs will not.

Multi Hypothesis Techniques

All the above approaches aim at a more precise modelling of the uncertainty during state estimation and therefore at obtaining a "better" (more likely) state, but none of them can deal with fully ambiguous situations. The plain KF for example has been proven to be generally able to converge to the optimal solution from any initial guess if the sensor noise is not too large, but the EKF in this application can not do this because it needs a "correct" initial guess to establish the association of the landmarks. With no initial guess about the state all possible associations of landmarks have to be taken into account, some of which may – depending on the map – be ambiguous. This is actually equivalent to the *kidnapped robot problem*, in which the robot is manually moved without giving the localisation a chance to track that motion.

In these cases *multi hypothesis techniques* (MHT) have to be employed, for example tracking each of the possible states with a separate KF. These KFs are then evaluated during some exploratory motions until one of them becomes favourable. As such, MHT are similar to the SLAM problem which also needs exploratory motions. Since exploratory motions with no real knowledge about where the platform is going to do not comply with the laboratory task definition MHT and SLAM have not been used in this work. They remain, however, interesting for other applications.

3.5.3 Path-Planning

The current path planner is based on the assumption that all obstacles are static and known in advance, which is a fairly legal assumption for a laboratory which has to comply to several safety standards. The necessary map expansion and visibility and tangent graphs can therefore be computed in advance. The motions are executed strictly according to the paths with no additional obstacle avoidance behaviour. The only way to react to an unexpected obstacle (a human?) is therefore to stop and wait until it disappears, which may take quite a lot of time and impair the sample management.

Dynamic Maps

One way to deal with such a situation is to use *dynamic maps*, i.e. to allow addition and removal of obstacles. Once an unexpected obstacle has been encountered and the platform has come to a stop it can wait for some time and see if the obstacles goes away. If it does not, those parts of its shape that can be seen are added as a new obstacle to the map, the map computation is repeated and a new path is generated (which may of course be impossible because now the way to the target may be blocked). At the same time obstacles which are obviously no longer present are removed from the map.

In order to not cause problems by erroneously overlooking fixed obstacles like glass doors this motivates *multi layer maps* in which permanent and temporary obstacles are kept separated and it is not allowed to remove permanent ones. Other layers might be added for objects which may change, but not be removed, like doors. These layers are only needed to separate different types of obstacles, but for computing the graphs they have to be merged. Such a map modifier can be implemented by an additional thread which only monitors the sensor data and does not further interfere with existing code. The implementation is simplified by the fact that the cigar expansion does not make any assumptions about whether lines belong together, and so a line can be easily added or removed.

3.5.4 Behaviours

Controlling a mobile platform is not trivial. Despite all the research that is done with them they are still far from being everyday devices like TV sets or VCRs. It is only just now that the first autonomous mobile devices like the Dyson DC06 vacuum cleaner become publically available.[54] This often amazes the layman because the basic problem can be so easily described:

Go to XXX (on the shortest path) and do not crash into anything on your way.

The problem is, nobody can really translate this into something that is understandable by a machine, simply because nobody has yet fully understood how he/she itself does it. All that we know is that a human uses a lot of low-level behaviour for this - one does not have to compute the distance of oneself to obstacles and try and not let any of these become smaller than a fixed threshold, it simply works by trying to “keep away from obstacles”. The obvious idea is of course to transfer the human solution to the robot, to implement a few “behaviours” and let the robot deal with the problem on its own.

Using behaviour based control means to implement a set of rules on what to do if certain patterns of sensor data occur. Behaviours at this level are most often not more than low-level reflexes, and so the rules are usually simple and easy to implement. Yet, by evaluating a set set of low-level rules and superimposing their reactions high-level swarm, pack or team behaviours can emerge.

As a rather primitive example, collision avoidance (up to some degree) for a mobile platform with a differential drive can be achieved by implementing two very trivial rules:

1. If you see an obstacle on the right, steer a bit to the left.
2. If you see an obstacle on the left, steer a bit to the right.

The evaluation of these rules of course has to properly cooperate with whatever instance is generating the underlying motion, which would not be the case with the translation motion class that tries to come back to the desired trajectory as implemented in this work. Also, one has to be aware that such simple rules may raise other problems when used carelessly: If, for example, the platform is moving into a funnel where both rules fire to the same degree the result may be that it accelerates and crashes into the end of the funnel – depending on how “steer to the left/right” is implemented. The point is, the above rules say to steer out of the way of obstacles, but they do not explicitly say to stop if the obstacle comes too close. They really only implement some kind of rudimentary collision avoidance, but nothing more.

[54] See `http://www.international.dyson.com/range/feature_frame.asp?model=DC06`.

The obvious way out of this is of course to add more rules that deal with assuring that all the other requirements are also met. Each rule on its own is easy to express (and hopefully to implement), but the question of what the possible results of their superimposition might be is still not answered. A (primitive) complete motion control for a mobile platform may for example be implemented with the following rules:

1. If the target is “far” away and the platform is “not” heading towards it, produce a positive command for one motor and a negative one for the other, so that the platform rotates towards the target.
2. If the target is “far” away and the platform “is” heading towards it, produce equal positive commands for both motors, so that the platform moves towards the target.
3. If an obstacle appears on the right side ahead, reduce the command for the left wheel by some factor, so that the platform turns to the left.
4. If an obstacle appears on the left side ahead, reduce the command for the right wheel by some factor, so that the platform turns to the right.
5. If an obstacle comes too close to the platform, reduce the commands for both motors, so that the platform eventually stops completely.
6. If the target is “close” and the platform is “not” heading towards the target direction, produce a positive command for one motor and a negative one for the other, so that the platform rotates towards the target direction.
7. If none of the above rules has fired for some time interval consider the motion finished.

Note that the rules about steering out of the way of obstacles have been rewritten more clearly: The command now is to explicity steer by reducing the velocity of the wheel on that side where the platform is to steer to and increasing the opposite one. This way the platform will not accelerate in a funnel and avoid crashing into it, but still an additional rule implementing a safety stop is required because the collision avoidance rules only speak about “obstacles on the left/right side”, not directly ahead.

The original GENCONTROL software behaved similar to this, in particular it used a comparable way to circumvent obstacles, but no path planner. As a result, as has been mentioned, it can drive around some obstacles if that does not mean to leave the trajectory too much. But it can also get stuck in a deadlock if by avoiding obstacles it comes too much off the target direction.

Rule based behaviour control can be interesting for low-level reflexes or if the research of emergence is the primary goal. If the primary goal is to have stable and deterministic “behaviour” it is not suitable. The superimposed reaction can – by definition – often not be predicted, and a robot with an unpredictable and indeterministic behaviour is potentially dangerous and should not be used in environments where humans can be harmed. It is primarily because of these troubles that behaviours have not been researched in this work. For an introduction to the field of behaviour based robotics see [Arkin 1998].

3.6 Experimental Results

The basic definition of “success” of the mobile platform is only a boolean variable stating whether it did bring the arm to a position that allowed it to perform the required operations or

not, and the success rate a probability of how often it did. This, however, is not a meaningful measure as for all the tests of the software done for this evaluation it has been 1.0 – complete success. The problem is that this measure does not take into account how close to a potential failure each individual run has been, because it does not use a proper definition of "success" of the arm.

The "success" of the arm should also not be defined as a boolean variable because of the same reason: it is not a very helpful information. If a failure occurs it is essential to know why this happened, but with only a boolean measure about how "well-behaved" a situation is no conclusions can be drawn and no improvements be made. Other measures have to be used, based on more sophisticated analyses of how a failure is caused.

Even though failure as such is a boolean entity, a situation may become gradually troublesome before finally resulting in a failure. In case of the arm, objects and motions are designed to have a certain amount of tolerance against unforeseen errors. When inserting tubes into slots – also known as the *peg-in-hole* problem (see figure 3.51) – there is a certain safety margin, and in case of the 50 ml NUNC tubes even bevelled edges. The vision system (see chapter 5) is expected to compensate as much of these displacements as possible, but even with complete compensation some problems remain:

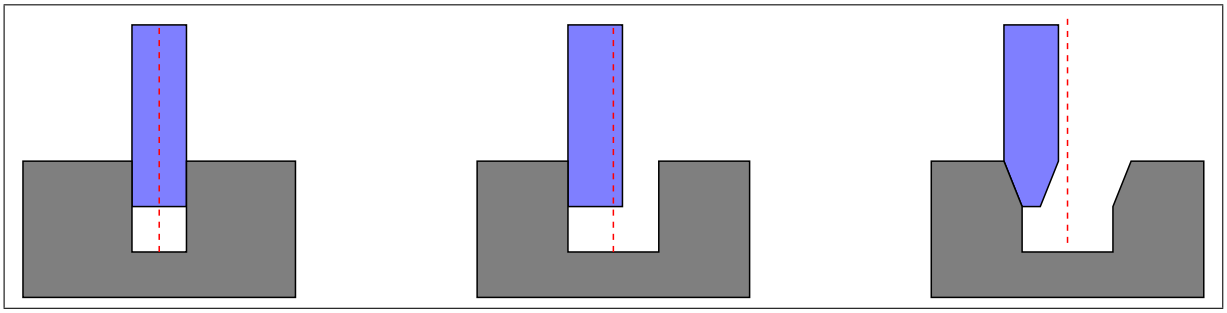


Figure 3.51: The peg-in-hole problem illustrating the need for safety margins: Apart from the optimal situation (left) a safety margin is needed to allow for unforeseen displacements (middle). In case of even larger errors bevelled edges (right) might be needed.

- First, the vision system may have been able to compensate the displacement by applying corrections to the robot position when it is centred over the feature that is used to detect that displacement, but this does not yet ensure that the subsequent operations will succeed. For example a small rotation of an object of a few degrees can easily be compensated if the object is standing fairly close to the arm, but leads to that large translations are needed for targets that are farer away. As a result the arm may run into workspace limits during the subsequent operation.

- Second, if the above problem does not occur, it may still happen that the motions during arm operation pass near a workspace boundary or singularity. In these cases the movement can become jerky, leading to that liquid from a filled tube can be spilled or the insertion of a tube into a slot can fail despite of all safety precautions. As a result the operation will fail even though it at first looked like everything was well-behaved.

At least the second effect is not a boolean, but a continuous one. The only way to guarantee success is to ensure that the real situation does not differ too much from the ideal situation,

say, that the deviation to be compensated by the vision is as small as possible. For the mobile platform this means that it should bring the arm as close to the intended position as possible, even though it may seem as there are several centimetres of safety margin. A safety margin exists to remain a safety margin, not to be used by default.

This means that – not too surprising – the quality of the mobile platform control software has to be measured as the precision with which it brings the platform to the intended target. Unfortunately, this is not as easy as it appears at first sight, as will be shown with the results of several test runs.

3.6.1 Localisation Position

The easiest value to analyse is the position as reported by the mobile platform's own localisation. For this, several experiments have been done, each one subsequently moving the mobile platform to one of a set of targets in a loop. For each target the position as reported by the localisation has been recorded. The plots of the these positions can be seen in figure 3.53 (the orientations have for simplicity been omitted).

The analysis of these plots reveals that the error can not solely be described by a gaussian distribution because it has some deterministic components. One of these components comes from the motion execution which can, as has been shown in section 3.4, overshoot. Another aspect that can be seen is that different experiments produce different results, and even in one experiment the different targets are reached with different precision. This reflects the fact that the accuracy of the localisation largely depends on the number and quality of observable landmarks, and the experiments and targets were deliberately chosen to represent some variety in these.

A detailed numeric analysis of these plots is given in table 3.4. Taking the highlighted worst case values from this table (Δx = -0.01268 m, Δy = 0.01455 mσ_x = 0.00672 and σ_y = 0.00434 – all values in meters) the accuracy of the mobile platform appears to be very good – less than 2 cm systematic deviation and less than 1 cm gaussian errors distribution around that. However, this measurement has a severe drawback.

experiment	# runs	# target	μ_x	y_0	Δx	Δy	σ_x	σ_y
20031023-1	119	1	1.99853	1.99725	-0.00147	-0.00275	0.00242	0.00266
	119	2	6.00322	4.00394	0.00322	0.00394	0.00237	0.00332
	119	3	1.99798	4.00277	-0.00202	0.00277	0.00232	0.00239
20031024	92	1	1.99870	1.99711	-0.00130	-0.00289	0.00340	0.00299
	92	2	6.00378	4.00440	0.00378	0.00440	0.00242	0.00316
	92	3	1.99661	4.00177	-0.00339	0.00177	0.00672	0.00207
20031216-d20	42	1	1.99510	2.00251	-0.00490	0.00251	0.00254	0.00238
	42	2	5.99825	1.99452	-0.00175	-0.00548	0.00260	0.00236
	42	3	5.98732	5.01455	-0.01268	0.01455	0.00434	0.00434
20031217-d20	46	1	1.99552	2.00302	-0.00448	0.00302	0.00227	0.00233
	46	2	5.99788	1.99415	-0.00212	-0.00585	0.00217	0.00198
	46	3	5.98992	5.01301	-0.01008	0.01301	0.00343	0.00384

Table 3.4: Mean values μ_x and μ_y, differences Δx and Δy of these from the target and standard deviations σ_x and σ_y of the distributions of the experiments (all values are in metres). The worst case values for each column have been highlighted.

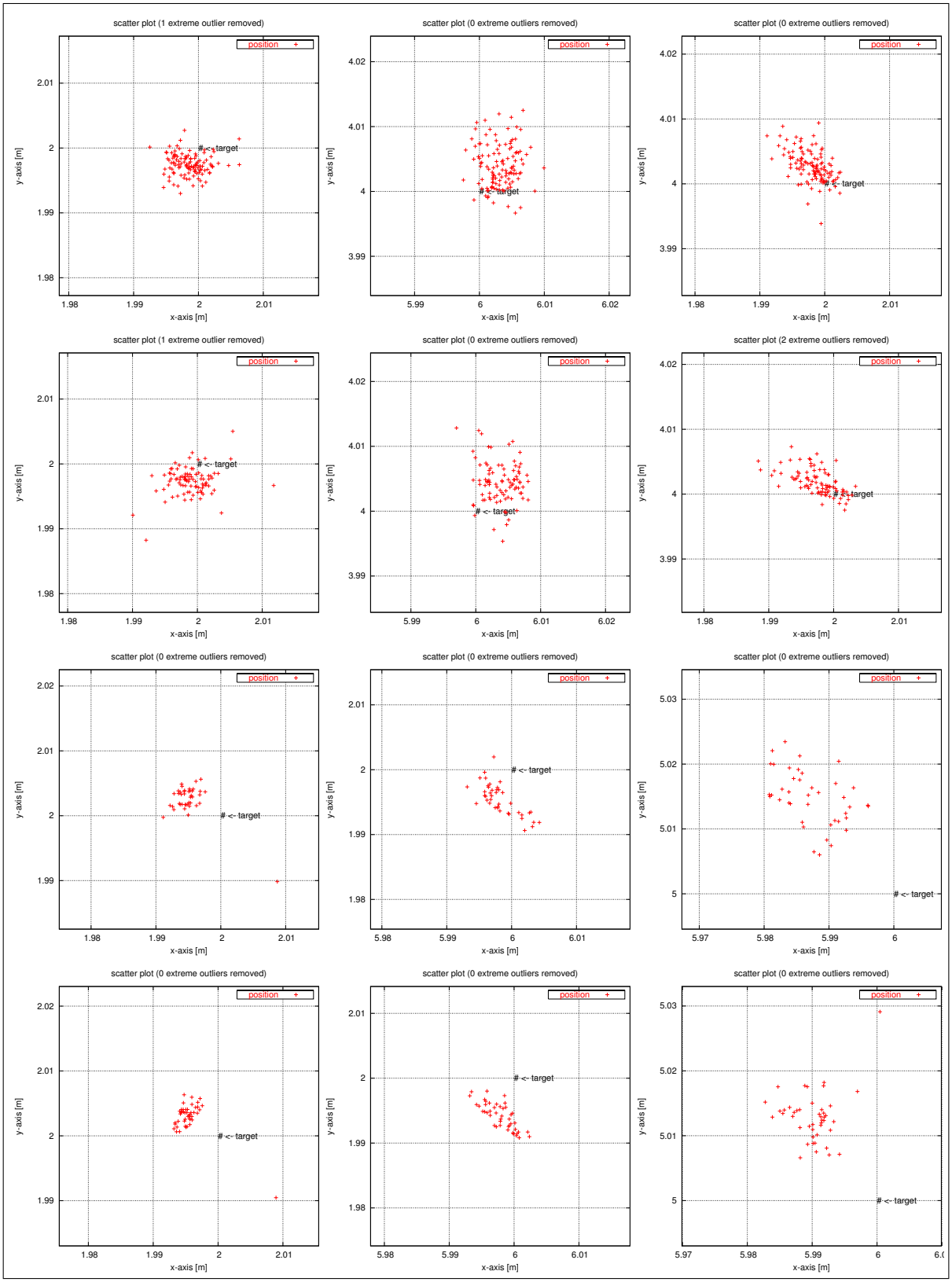


Figure 3.52: Plots of the position as reported by the localisation. Experiments shown in rows from top to bottom: 20031023-1, 20031024, 20031216-d20 and 20031217-d20. Each experiment consists of 3 targets, which are shown in the columns.

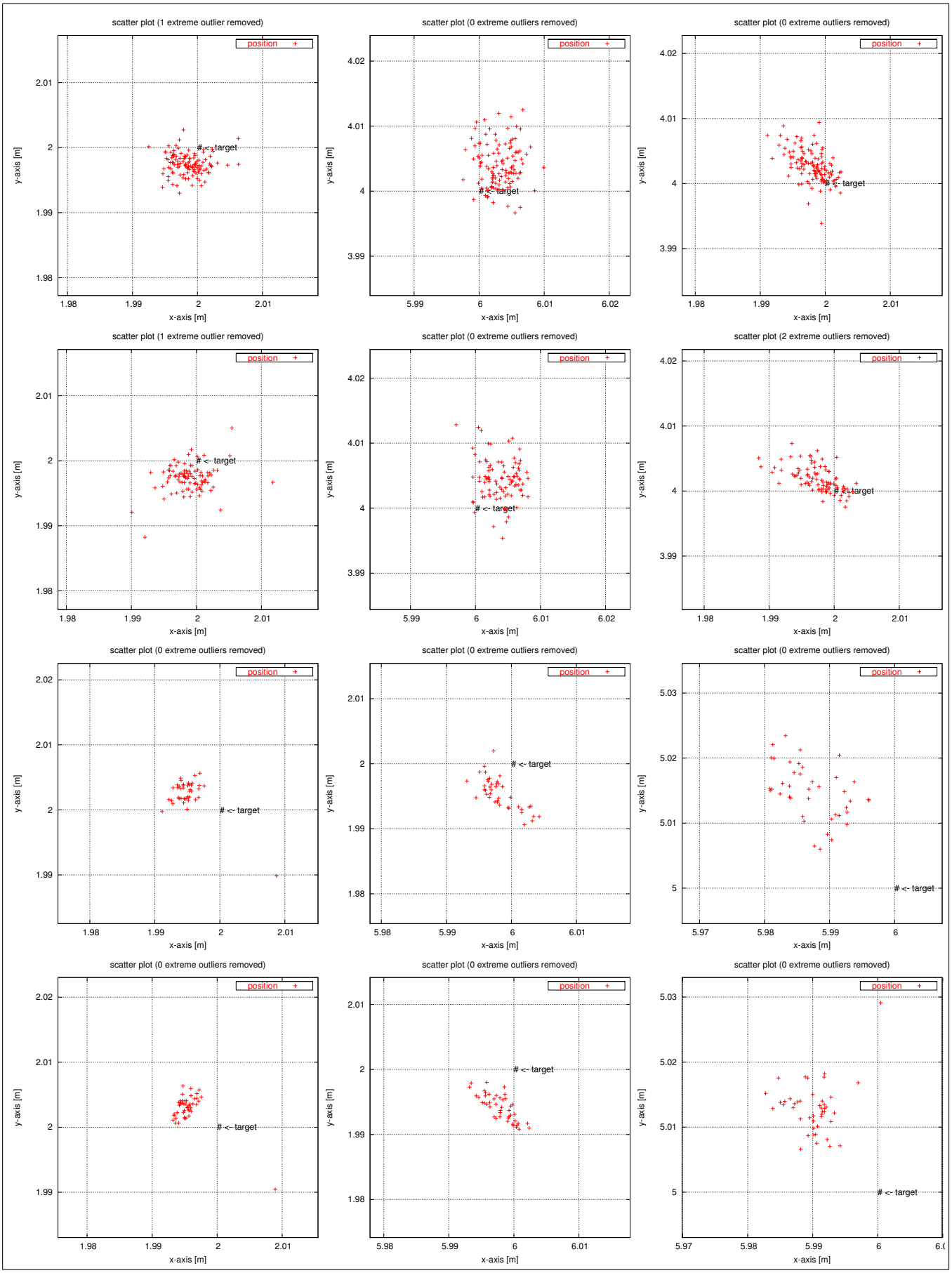


Figure 3.53: Plots of the position as reported by the localisation. Experiments shown in rows from top to bottom: 20031023-1, 20031024, 20031216-d20 and 20031217-d20. Each experiment consists of 3 targets, which are shown in the columns.

The problem is that this measurement expresses only the localisation's *belief* about where it has been, not the real position in the real world. This belief includes all errors of the EKF used for the localisation, all errors in detecting and measuring the landmarks and all errors that result from an initial misplacement or unevenly spreading of landmarks. It can happen that the platform is at a completely wrong position, but the localisation believes the position is correct because of these errors. Analysing only the localisation's position output means to analyse a *guess* only, with no checking against an absolute reference. As a result, this measurement can not be used to judge the accuracy of the mobile platform.

3.6.2 Absolute Verification of Localisation Position

Schneider and Westhoff therefore use an absolute reference against which they check the localisation's position[55]. They attach measuring rods to the platform as in figure 3.54 and place a millimetre grid around the single target position (x=0 m, y=0 m and φ=0 °). The platform is then commanded to repeatedly drive to that target in a motion along the x-axis and both the localisation's position and the real position as computed from the position of the rods is recorded once it has reached the target.

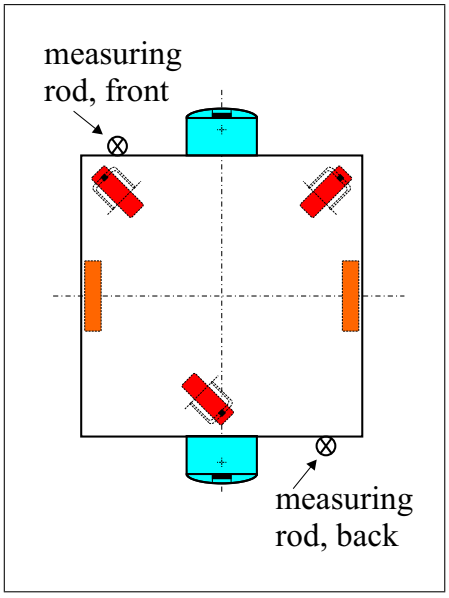


Figure 3.54: Measurement rods at the mobile platform.

This test was originally used to demonstrate and document the performance differences between the old *GenControl* software and (an older version of) the software developed in this work (called *UniBiControl* here), leading to two sets of data in the following figures.

Figure 3.55 shows the localisation's estimate of the position. Since the motion was along the x-axis it appears as if GenControl stops too early. Also, the orientation seems to be much better with UniBiControl than with GenControl.

In comparison to this, figure 3.56 shows the real position, obtained using the measuring rods. The impression that GenControl stops too early and is not as good in the orientation as UniBiControl can be verified with this measurement.

[55] See [Schneider and Westhoff 2002].

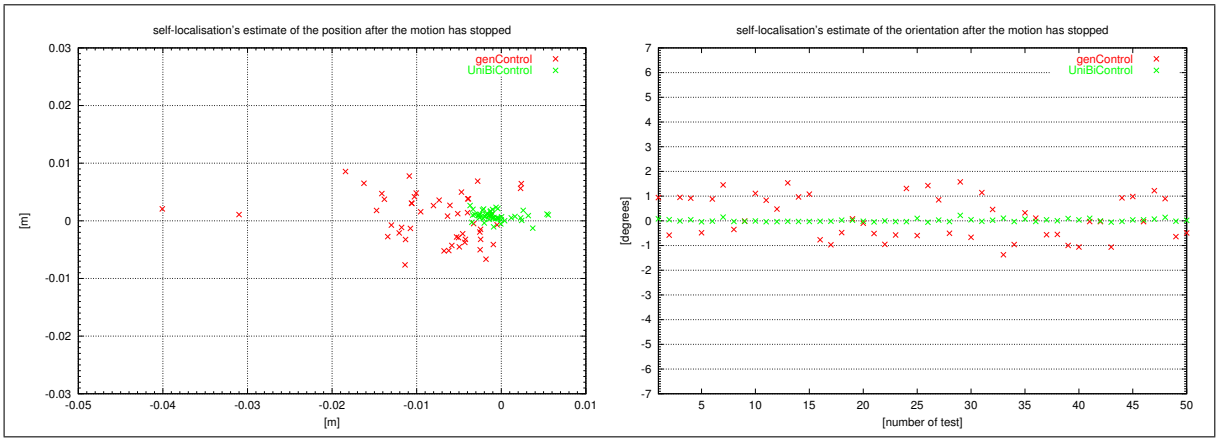


Figure 3.55: The mobile platform position as reported by the localisation.

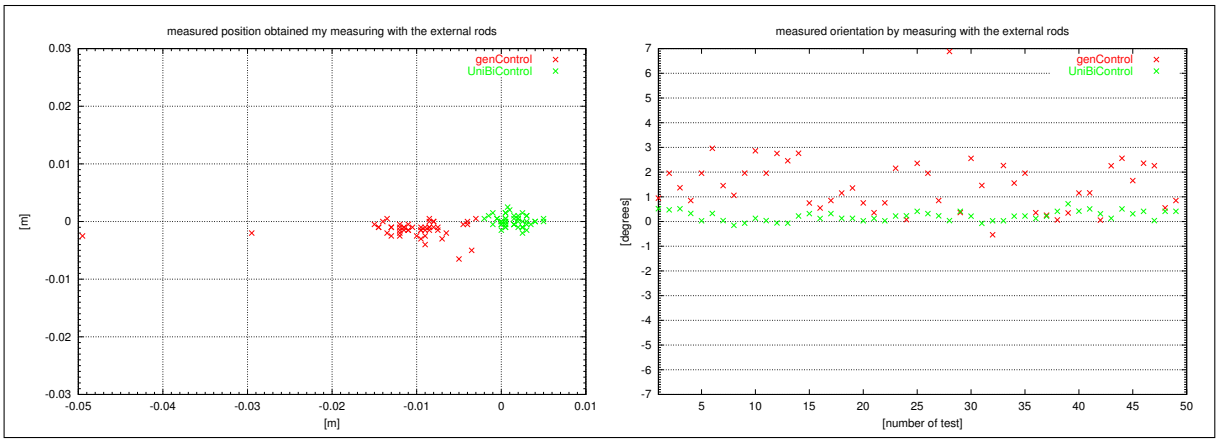


Figure 3.56: The mobile platform position as measured with the measurement rods.

Numerically speaking, the parameters of the distributions are as in table 3.5. As can be seen, the differences in the μ values are 2 mm in x- and 1.8 mm in y-direction for GenControl, and 0.5 mm and 0.7 mm for UniBiControl respectively. These rather small differences mean that – on average – both control softwares are not too bad in estimating the real position.

	localisation		verification	
	GenControl	UniBiControl	GenControl	UniBiControl
$\max x$ [mm]	-40.1	5.6		
$\max y$ [mm]	8.6	2.7		
$\max \varphi$ [°]	1.57	0.22		
μ_x [mm]	-8.1	-0.7	-10.1	-1.2
μ_y [mm]	0.4	0.8	-1.4	0.1
$\mu\varphi$ [°]	0.14	0.02	1.58	0.23
σ_x [mm]	7.5	2.2	6.9	1.6
σ_y [mm]	4.1	0.8	1.3	0.9
$\sigma\varphi$ [°]	0.086	0.006	1.9	0.3

Table 3.5: Parameters of the distributions in figures 3.55 and 3.56.

The σ values for GenControl are a bit lower for the real than for the estimated position, whereas for UniBiControl they are roughly equal. This means that the noise of the position in GenControl is larger than in UniBiControl.

Overall, the measurements attest that GenControl does show considerable room for improvements. In particular the average systematic x-deviation of about -1 cm (in this experiment!) shows that the questions of accuracy of the localisation and of the motion execution are indeed separate. It is mainly because of the "accuracy" of the motion execution of GenControl that the decision to replace it was taken in the first place.

Concerning the type of measurement it can be stated that because of the reference to an absolute frame it has to be taken as highly accurate and reliable. The practical realisation, however, is complex, expensive and potentially troublesome. Since the readings have to be taken from the measuring rods manually all possible human errors have to be considered.

As a result of this, this measurement has not been repeated for later versions of UniBiControl. In particular, it has not been applied to situations with multiple targets that are placed so that different numbers of landmarks can be seen from them. Instead, an approach that automates the measurement is used.

3.6.3 Relative Visual Verification of Localisation Position

The basic desire behind the automation is that one of the already existing sensors on the mobile robot should be used for the necessary measurements. With laser scanners, odometry and gyro compass already in use by the localisation this leaves only the camera to provide suitable information.

The camera can be used to detect marker on the floor free of systematical noise and with only very low statistical noise. As such it can provide a very reliable, accurate and repeatable measurement. In fact the taking of images and their processing to obtain the desired measurements can even be decoupled so that the analysis can be made off-line. The only problem lies in which kind of markers to use to obtain which kind of information.

Using any type of absolute marker grid is not a suitable idea because this grid would have to be too carefully placed in order for it to be guaranteed to align to the world coordinate system. If this alignment can not be established the displacement will turn up in the measurements as an appearingly systematical drive error and can never be parted from other systematical effects[56]. Fortunately, such an absolute measurement is not necessary to establish the basic accuracies.

Motivation

Instead, a relative measurement that only computes displacements between two images can be used, which allows to use the very same vision routines which are used and explained in chapter 5. Because of this, the measurement is called "*relative visual verification*" of the localisation position. The procedure is as follows:

- First, the camera is mounted on the platform rather than on the arm in such a way that it can see a part of the floor with a sufficient resolution.

[56] In fact it is not totally clear if/how [Schneider and Westhoff 2002] have considered this problem, the likeliest explanation being that they considered the displacement irrelevant until it showed up in the results with at least the magnitude of the basic accuracy, which it does not.

- Next, the platform is driven to the real target and a marker placed on the floor so that it is roughly in the centre of the camera image. Note that at this stage neither the real absolute position of the platform nor the real position of the marker is known.

- Then the platform is driven to the real target n times, the position p_i (of the camera, not the platform centre) as reported by the localisation is recorded and an image b_i taken, yielding a set of positions and a set of corresponding images.

After these preparations the actual evaluation can be done off-line:

- The visual displacement Δb_i between reference image b_1 and image b_i for $i = 1..n$ is computed, yielding a set of displacements instead of a set of images. Note that in this set the displacement for the reference image b_1 is 0, but neither this nor the number of the reference image in the first place does affect the further computation.

- From this set of displacements the average[57] visual displacement $\mu_{\Delta b}$ is computed. Since no statement can be made about the character of the reference situation no statement can be made about this $\mu_{\Delta b}$, in particular not if it is 0 or not.

- In addition to that, the average position μ_p is computed from the set of p_i.

Now, instead of taking the real target as a basis for further computations, the *virtual target* given by μ_p is considered. This assumes that *on average* this is identical to the real target, which allows only for statistical and not for systematical drive errors. If systematical drive errors occur, e.g. the platform always drives 10 cm too far, they are not detected by this measurement. However, once the (statistical) driving accuracy and measuring accuracy have been computed the absolute position measurements can be evaluated again and a systematical error can be detected.

- The *positional displacement* $\Delta p_i = p_i - \mu_p$ is then computed. This Δp_i is the deviation as seen by the localisation and is the overlay of the (pure statistical) drive error and the measurement error.

- The *visual displacement* $\Delta b'_i = \Delta b_i - \mu_{\Delta b}$ is computed next. This $\Delta b'_i$ is the deviation as seen by the camera and corresponds to the drive error only.

Both distributions Δp_i and $\Delta b'_i$ are by construction centred around zero. $\Delta b'_i$ corresponds to the drive error and $\Delta p_i - \Delta b'_i$ corresponds to the measurement error. A separation of the two errors is therefore possible.

The standard deviations σ of all distributions are then computed (expecting that $\sigma_{\Delta p_i} > \sigma_{\Delta b'_i}$). Since the $x : y$ distributions representing the translational error cannot be assured to be of circular shape a *principal component analysis* (PCA) is applied to them, leading to two *eigenvalues*. The square root of these eigenvalues corresponds to the standard deviation σ^{rot} of a new distribution rotated appropriately so that the largest extension lies in the new x-direction. Because the σ^{rot} are therefore more sensitive to worst case elements they are generally larger than the plain σ. It is these σ^{rot} that are taken to judge the "accuracy" of the system.

[57] In case of extreme outliers in the distribution the *median* may be used instead of the *mean* for this averaging.

Experiments

Again, several experiments have been done and evaluated accordingly, the results for the translational errors being shown in figures 3.58 to 3.62 on the following pages. In these figures the Δp_i distributions giving the localisation error have been labelled as "LOC", the $\Delta b'_i$ distributions giving the vision (or positioning) error as "VIS" and the $\Delta p_i - \Delta b'_i$ distributions giving their difference (or measuring) error as "DIFF".

Each of these experiments (figures) consists of plots for three different targets, each in a separate row. The left column shows a scatter plot of the LOC, VIS and DIFF distributions as well as the *standard deviation ellipse* obtained from the PCA, the (non-square-rooted) eigenvalues of which are given in the legend. The right column shows correlation plots between the LOC and VIS distributions of the x and y axis, giving the correlation coefficients ϱ in the legend.

A numerical evaluation of these plots is given in table 3.6, using the same abbreviations LOC, VIS and DIFF as in the figures. In this table each row gives the values for one distribution of one target of one experiment. The first two columns give the extrema $\|\Delta\vec{p}\|_{\max}$ and $\|\Delta\varphi\|_{\max}$. The next columns give the standard deviations σ_x, σ_y and σ_φ of the distribution and the square-rooted eigenvalues σ_x^{rot} and σ_y^{rot} of the PCA. The *absolute value* of $\sigma^{\mathrm{rot}} = \sqrt{(\sigma_x^{\mathrm{rot}^2} + \sigma_y^{\mathrm{rot}^2})}$ can be used as "the" accuracy. The remaining columns are specific for the target as a whole, and not for one distribution (though listed under the LOC row). They give the correlation coefficients ϱ_x and ϱ_y of the LOC vs. VIS distributions (which is expected to be partly correlated) and of the DIFF vs. VIS distribution (which is expected to be not correlated). Finally, the corresponding ϱ_φ for the orientation are given.

In most cases the order of magnitude of the corresponding values are similar, however, a few measurements stand out:

- In the 3rd target of experiment 20031022-2 the σ^{rot} is 12.287 mm for LOC and 11.393 mm for VIS, but only 3.225 mm for DIFF. This means that even though the measuring accuracy is pretty good the localisation has reported large errors, which can also be seen from the fact this case produces the largest extrema. The reason for this is of course that the motion execution has produced rather large positioning errors. This is also confirmed by the vision, which reports a high correlation of 0.973 and 0.975 between LOC and VIS data, indicating agreeing measurements. The correlation between DIFF and VIS data is – expectedly – much lower.

- In a lot of other cases the correlation between LOC and VIS data is also very low, or rather non-existent, down to even negative. This is particularly true for cases where the σ values are already either small or similar for LOC, VIS and DIFF distributions and means that no pure positioning error can be isolated from the overlayed localisation error. Building DIFF on the contrary introduces a systematic component and therefore leads to a higher (negative) DIFF/VIS correlation than LOC/VIS.

- For the experiments 20031022-2, 20031216-d20 and 20031217-d20 the values for the third target are significantly higher than for the first and second target. This – as figure 3.57 shows – is because the third target is in an area of the map where fewer landmarks can be seen, and those that can be seen are farther away. As a result the measuring accuracy decreases, and as a result from this the positioning accuracy decreases too. This effect can also be very clearly seen in the scatter plots.

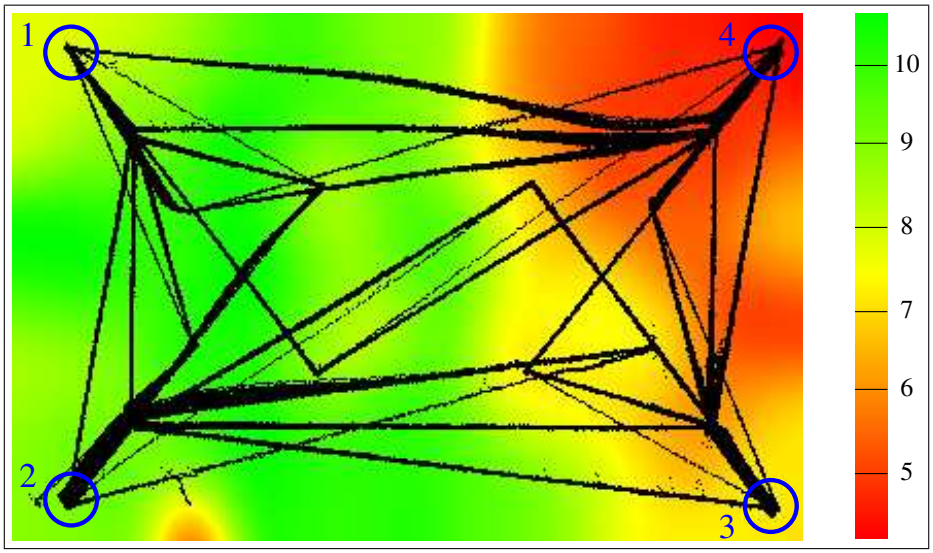


Figure 3.57: Average number of visible reflector marks.
Target #4 has not been used in the experiments.

The measuring accuracy DIFF (σ^{rot}) ranges from 1.999 to 6.175 mm with an average of 3.37 mm, the positioning accuracy VIS ranges from 3.070 to 11.393 mm with an average of 4.761 mm and the localisation accuracy LOC ranges from 2.922 to 12.287 mm with an average of 4.650 mm. The fact that the average values are pretty similar indicates that under normal circumstances the measuring noise determines the global accuracy and the motion execution has only a negligible effect. On the other hand, in those cases where larger localisation errors can be observed they correlate with larger vision errors, indicating that errors of the motion execution can be safely detected. In any case the accuracy is well within the requirements of this work.

The results for the orientational errors are shown in figures 3.63 to 3.67 on the following pages, using the same abbreviations as above. The results do not seem to be spectacular with 0.804 being the highest correlation between LOC and VIS values, suggesting that even less gain can be achieved by trying to compensate for localisation errors. This is not too surprising, taking into account that the motion execution issues a rotation to the target orientation as the last step. As a result of this the orientation is of course expected to be more accurate than the position, which may change a little but during this final rotation.

This is also confirmed by LOC $\sigma\varphi$ values in the range of 0.015 to 0.111°, with an average of 0.0518°. In fact even the LOC $\|\Delta\varphi\|_{\max}$ are only in a range of 0.03 to 0.32° with an average of 0.163°, and the target VIS $\|\Delta\varphi\|_{\max}$ is only 0.42°. With this accuracy even the largest outliers can simply be ignored for the requirements of this work.

3.6.4 Summary

Overall, the σ^{rot} values for LOC confirms that the mobile platform clearly has an accuracy of "less than 1 cm", as has been published[58]. Also, the assumption that the measuring accuracy is better than the localisation accuracy can – on average – be confirmed. As a practical upshot of the often lacking correlation of LOC/VIS values the idea whether it is a good idea to compensate

[58] See [Scherer et. al. 2003].

for localisation errors has to be doubted. Except from a few cases where large positioning errors have occurred it is likely to yield no gain – on the contrary, it can make the situation even more troublesome. In a future version this code should therefore be replaced in favour of a repetition of the platform's approach motion in case the localisation should detect abnormally large errors (3 to 4 cm and above).

As far as the orientation is concerned the accuracy of less than 1/10th of a degree is very unlikely to be improvable. This accuracy does not mean that the possible error is bound to it, as the EKF can completely fail due to loosing the association of the landmarks as described in section 3.2.8, but only that this effect did not occur during these tests.

experiment	#targ	#runs		$\|\Delta\vec{p}\|_{\max}$ [mm]	$\|\Delta\varphi\|_{\max}$ [°]	σ_x [mm]	σ_y [mm]	σ_φ [°]	σ_x^{rot} [mm]	σ_y^{rot} [mm]	σ^{rot} [mm]	loc / vis ϱ_x [1]	loc / vis ϱ_y [1]	diff / vis ϱ_x [1]	diff / vis ϱ_y [1]	loc / vis ϱ_φ [1]	diff / vis ϱ_φ [1]
20031022-2	1	35	LOC	15.6	0.1	3.17	2.62	0.06	3.444	2.243	4.110	0.289	0.822	-0.350	-0.089	0.039	-0.791
			VIS	7.3	0.2	2.05	2.29	0.08	2.365	1.958	3.070						
			DIFF	13.2	0.3	3.23	1.50	0.09	3.456	0.876	3.565						
	2	35	LOC	7.6	0.3	3.50	2.59	0.11	3.713	2.267	4.350	0.339	0.508	-0.768	-0.759	0.682	0.047
			VIS	20.6	0.2	5.14	3.91	0.07	5.515	3.359	6.457						
			DIFF	19.6	0.2	5.14	3.42	0.08	5.167	3.381	6.175						
	3	35	LOC	43.5	0.2	9.63	7.63	0.09	9.683	7.563	12.287	0.973	0.975	0.540	-0.399	0.169	-0.781
			VIS	48.0	0.4	7.94	8.17	0.12	9.202	6.718	11.393						
			DIFF	11.4	0.4	2.65	1.84	0.14	2.660	1.824	3.225						
20031023-1	1	119	LOC	22.8	0.2	2.61	2.48	0.05	2.716	2.361	3.599	0.835	0.866	-0.284	-0.351	0.447	-0.785
			VIS	22.3	0.3	2.60	2.61	0.08	2.611	2.603	3.687						
			DIFF	9.7	0.3	1.50	1.32	0.07	1.565	1.242	1.998						
	2	119	LOC	9.3	0.3	3.35	2.51	0.08	3.348	2.504	4.181	0.381	0.498	-0.598	-0.476	0.791	-0.311
			VIS	10.8	0.3	3.58	2.43	0.08	3.693	2.250	4.324						
			DIFF	10.2	0.2	3.86	2.47	0.05	3.859	2.471	4.582						
	3	119	LOC	9.5	0.2	2.26	2.50	0.05	2.911	1.696	3.369	0.372	0.624	-0.588	-0.288	0.667	-0.695
			VIS	11.7	0.2	2.37	2.14	0.07	2.610	1.844	3.196						
			DIFF	11.2	0.2	2.60	2.04	0.05	2.603	2.028	3.300						
20031024	1	92	LOC	25.4	0.1	2.99	3.42	0.03	3.914	2.303	4.541	0.866	0.679	-0.013	-0.089	0.173	-0.947
			VIS	17.3	0.4	2.61	2.55	0.09	2.724	2.421	3.645						
			DIFF	13.8	0.3	1.49	2.52	0.09	2.668	1.211	2.930						
	2	92	LOC	10.6	0.2	3.11	2.46	0.06	3.145	2.411	3.963	0.484	0.440	-0.534	-0.636	0.681	-0.646
			VIS	18.4	0.3	3.22	2.90	0.08	3.609	2.400	4.334						
			DIFF	20.7	0.3	3.22	2.86	0.06	3.243	2.830	4.304						
	3	90	LOC	30.6	0.1	4.38	1.94	0.04	4.434	1.819	4.793	0.905	0.391	-0.208	-0.527	0.412	-0.839
			VIS	28.7	0.4	4.36	1.86	0.08	4.393	1.788	4.743						
			DIFF	8.1	0.4	1.91	2.10	0.07	2.130	1.875	2.838						
20031216-d20	1	42	LOC	18.5	0.1	1.61	3.09	0.02	3.092	1.607	3.485	0.288	0.911	-0.702	-0.175	0.364	-0.731
			VIS	17.6	0.1	1.99	3.04	0.03	3.040	1.988	3.632						
			DIFF	5.3	0.1	2.17	1.29	0.03	2.476	0.497	2.525						
	2	42	LOC	7.0	0.1	3.29	1.32	0.02	3.295	1.301	3.543	0.888	0.035	-0.020	-0.822	0.542	-0.898
			VIS	7.3	0.1	2.95	1.96	0.05	2.990	1.891	3.538						
			DIFF	10.9	0.1	1.51	2.32	0.04	2.336	1.492	2.772						
	3	42	LOC	10.3	0.2	3.27	5.24	0.07	5.244	3.274	6.182	0.858	0.900	0.086	-0.564	0.257	-0.731
			VIS	12.1	0.2	2.66	6.28	0.09	6.288	2.641	6.820						
			DIFF	6.4	0.2	1.69	2.77	0.10	2.814	1.608	3.241						
20031217-d20	1	46	LOC	18.2	0.1	1.48	2.85	0.03	2.851	1.479	3.212	0.312	0.916	-0.785	-0.290	0.191	-0.740
			VIS	19.9	0.1	2.24	2.96	0.03	3.089	2.059	3.712						
			DIFF	7.9	0.1	2.27	1.20	0.04	2.418	0.857	2.565						
	2	46	LOC	5.9	0.0	2.74	1.01	0.01	2.746	1.000	2.922	0.772	0.575	-0.362	-0.834	0.657	-0.976
			VIS	6.7	0.1	2.79	1.83	0.06	2.837	1.765	3.341						
			DIFF	5.4	0.1	1.87	1.50	0.05	2.208	0.928	2.395						
	3	46	LOC	19.6	0.1	3.86	3.50	0.04	3.974	3.371	5.211	0.767	0.696	-0.043	-0.650	0.530	-0.883
			VIS	18.3	0.2	3.07	4.59	0.09	4.726	2.851	5.519						
			DIFF	14.0	0.2	2.48	3.31	0.08	3.413	2.334	4.135						

Table 3.6: Results for the localisation (LOC), positioning (VIS) and measuring accuracy (DIFF). For explanations see page 120.

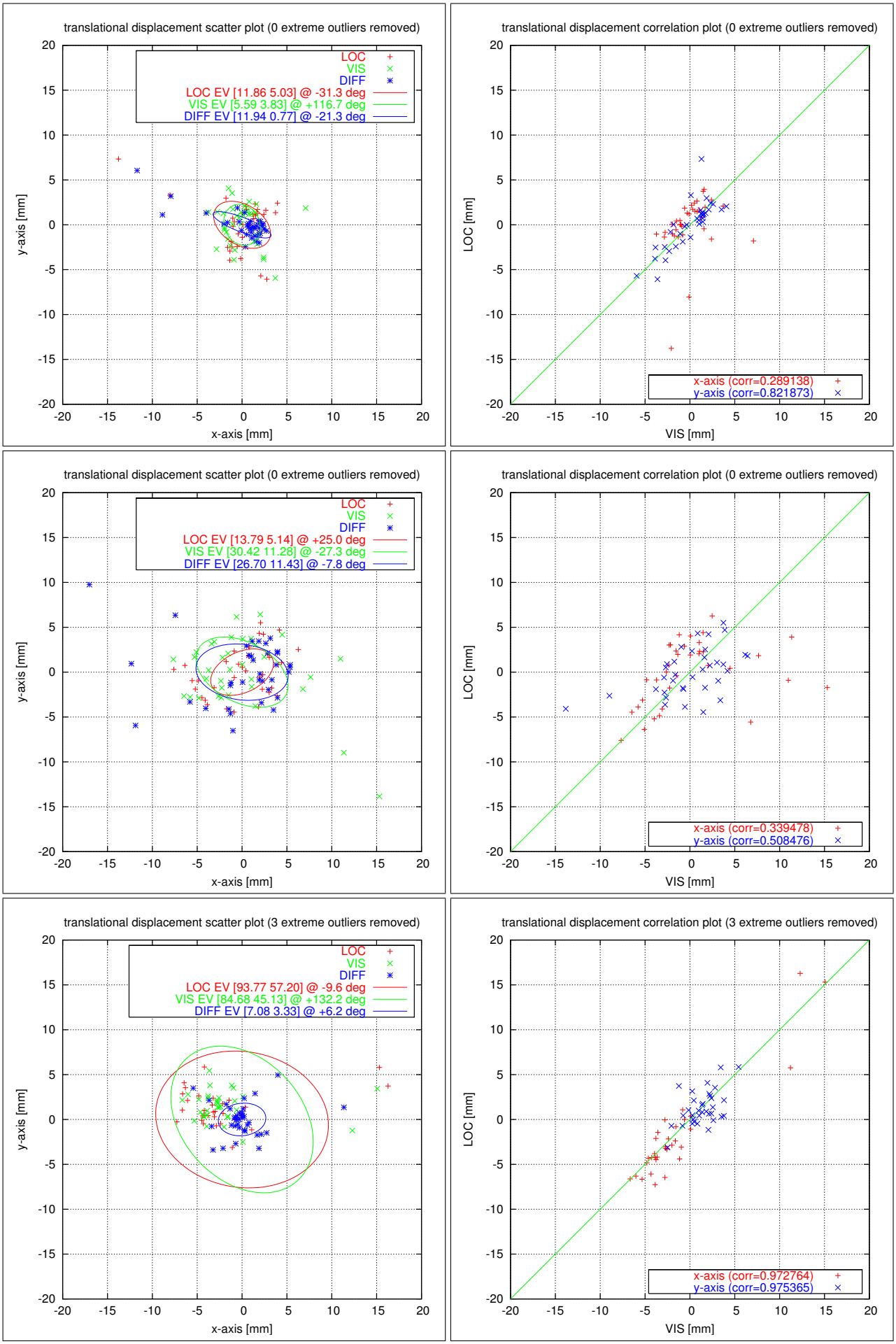


Figure 3.58: Experiment 20031022-2: Scatter and correlation plots of translational displacements for three different targets, each in a separate row. See page 120 for an explanation of the legend.

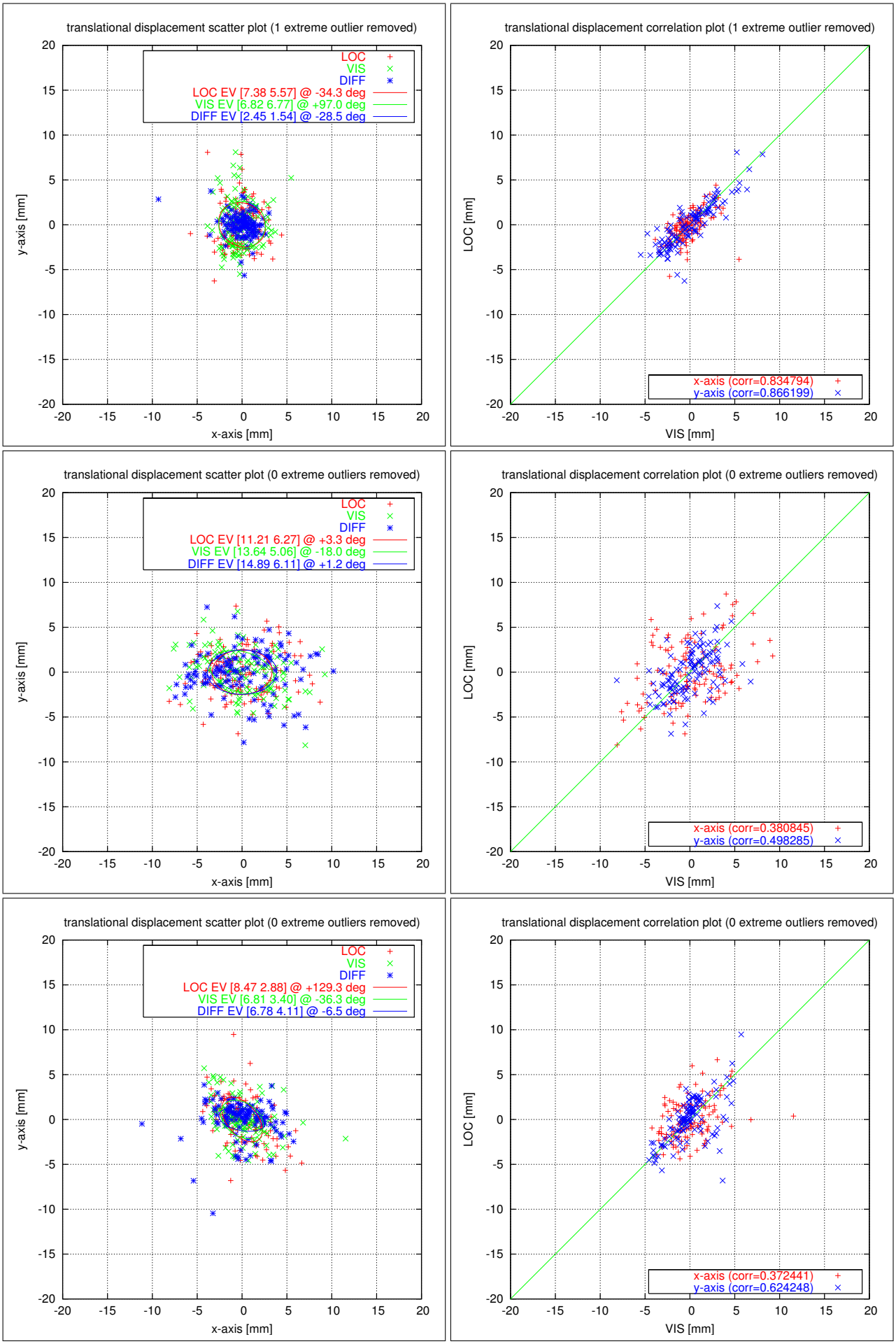


Figure 3.59: Experiment 20031023-1: Scatter and correlation plots of translational displacements for three different targets, each in a separate row. See page 120 for an explanation of the legend.

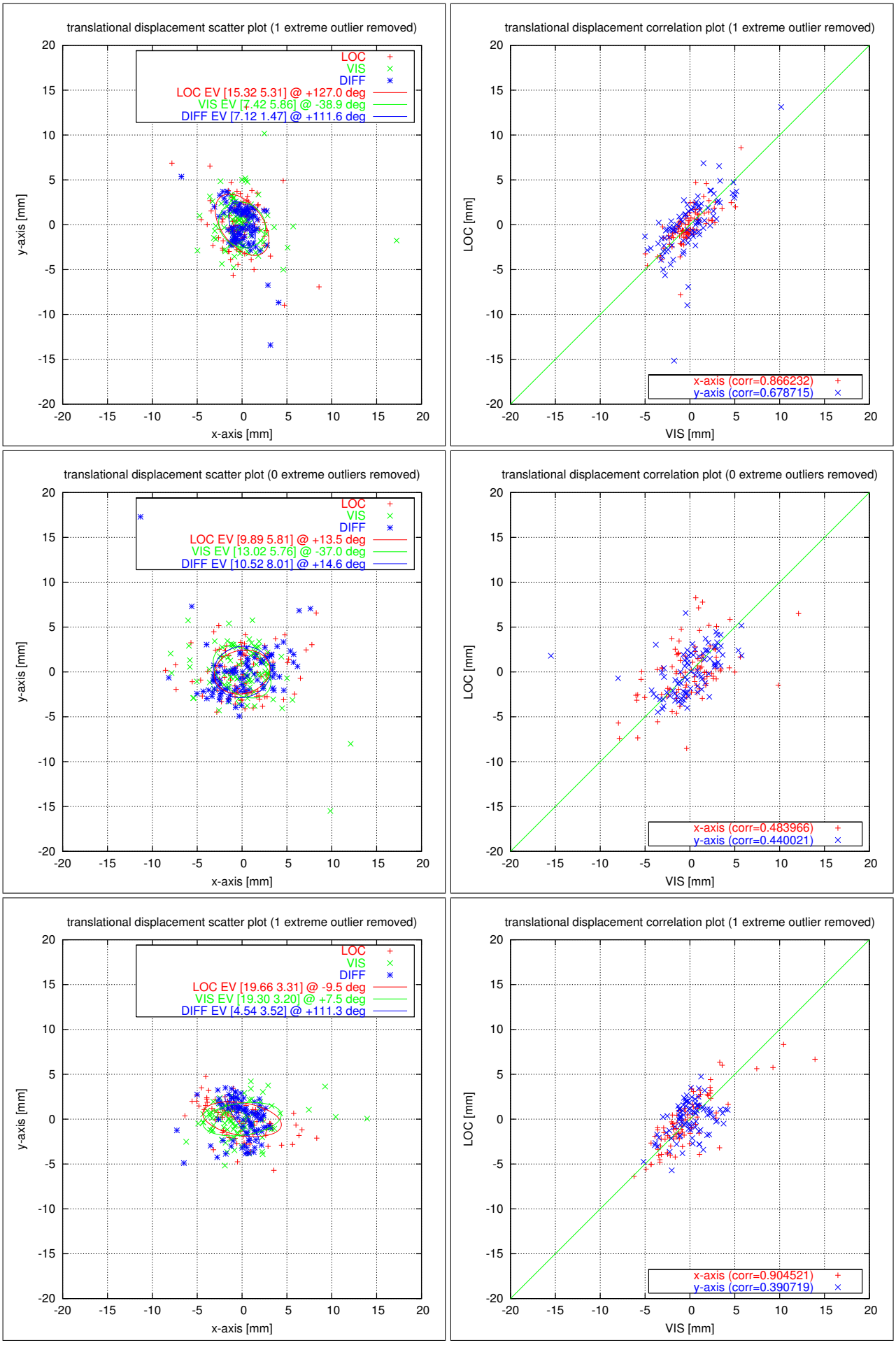


Figure 3.60: Experiment 20031024: Scatter and correlation plots of translational displacements for three different targets, each in a separate row. See page 120 for an explanation of the legend.

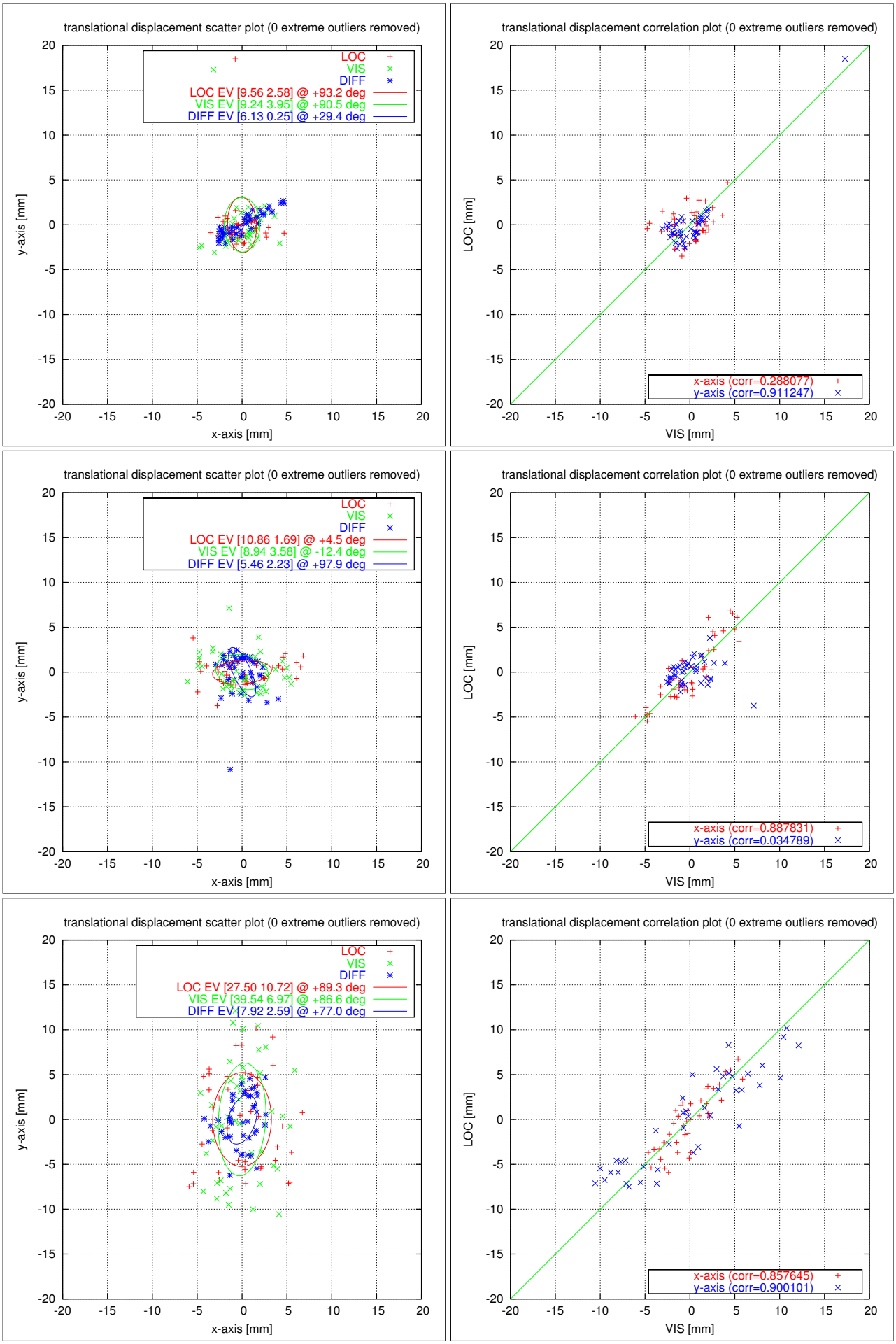


Figure 3.61: Experiment 20031216-d20: Scatter and correlation plots of translational displacements for three different targets, each in a separate row. See page 120 for an explanation of the legend.

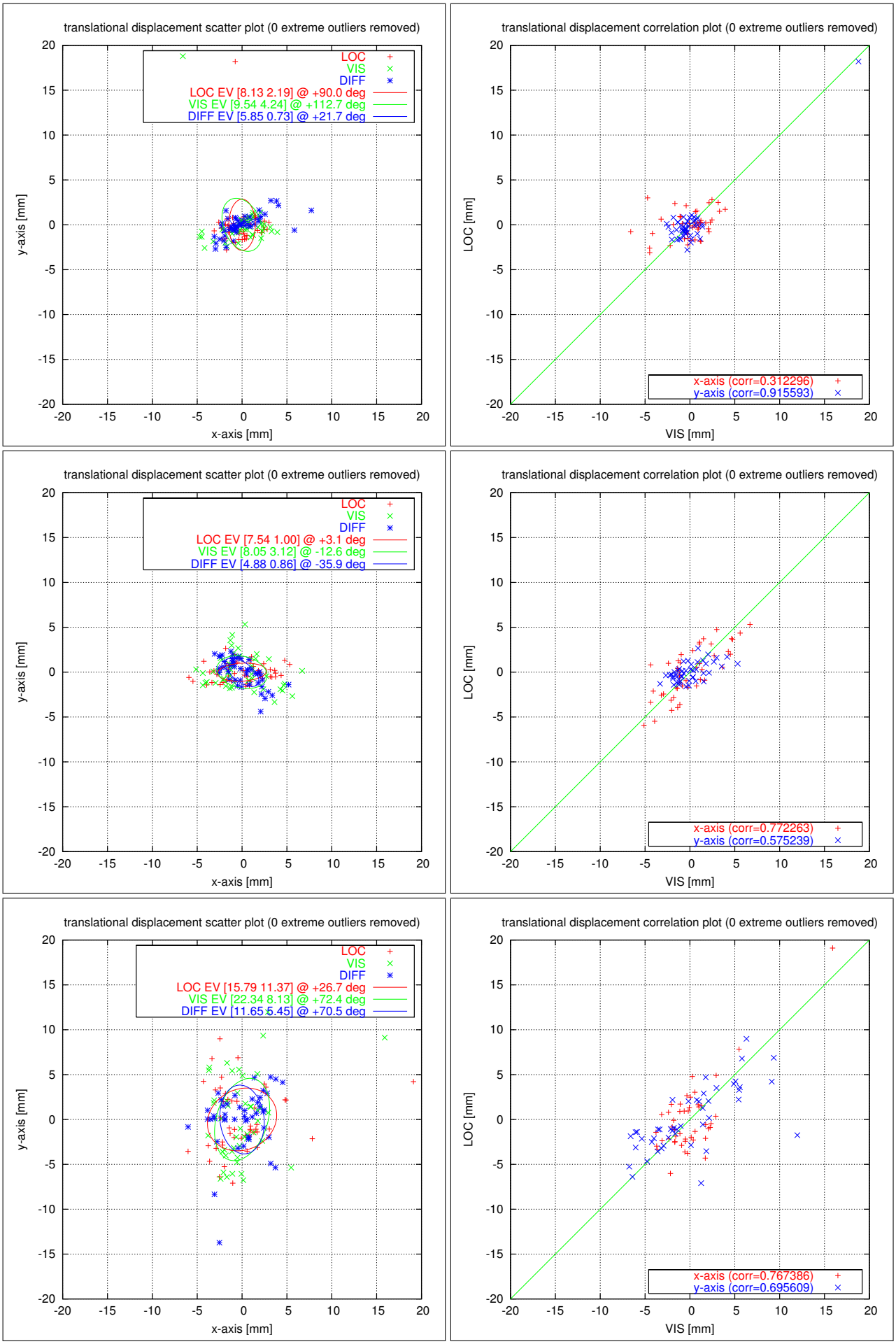


Figure 3.62: Experiment 20031217-d20: Scatter and correlation plots of translational displacements for three different targets, each in a separate row. See page 120 for an explanation of the legend.

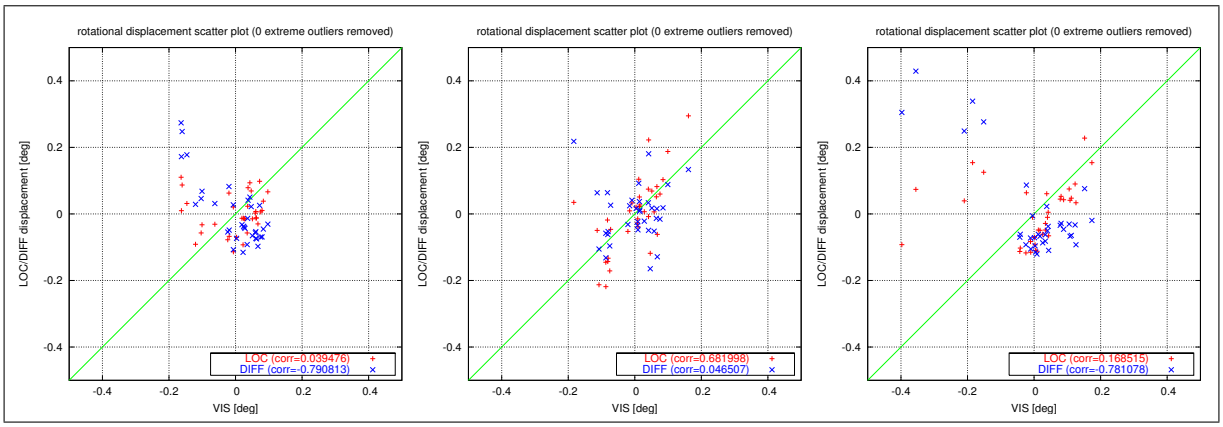


Figure 3.63: Experiment 20031022-2: Rotational displacements for three different targets. See page 121 for an explanation of the legend.

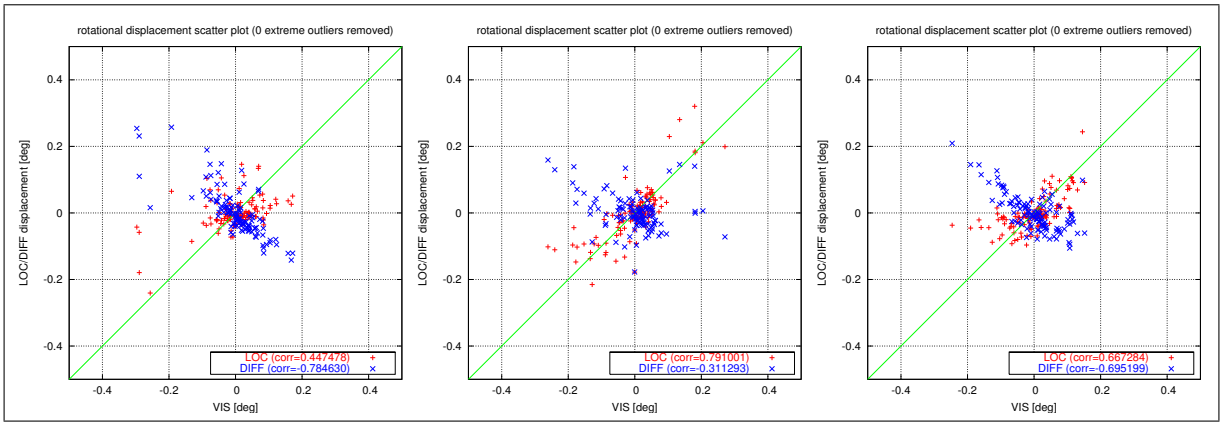


Figure 3.64: Experiment 20031023-1: Rotational displacements for three different targets. See page 121 for an explanation of the legend.

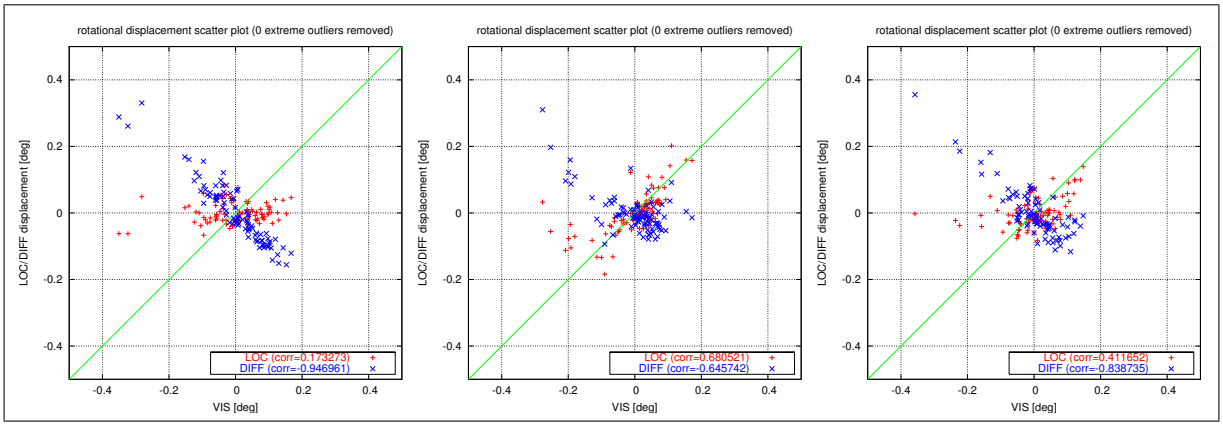


Figure 3.65: Experiment 20031024: Rotational displacements for three different targets. See page 121 for an explanation of the legend.

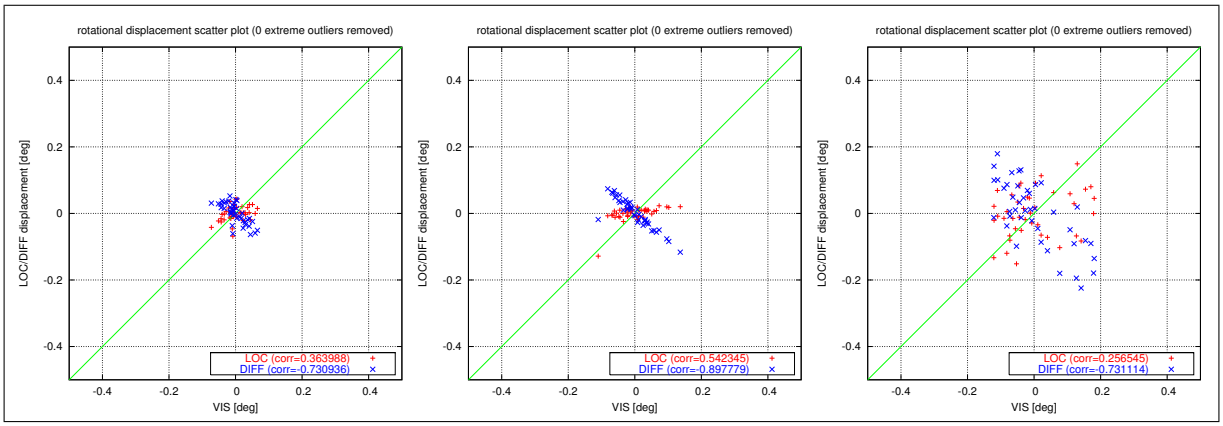


Figure 3.66: Experiment 20031216-d20: Rotational displacements for three different targets. See page 121 for an explanation of the legend.

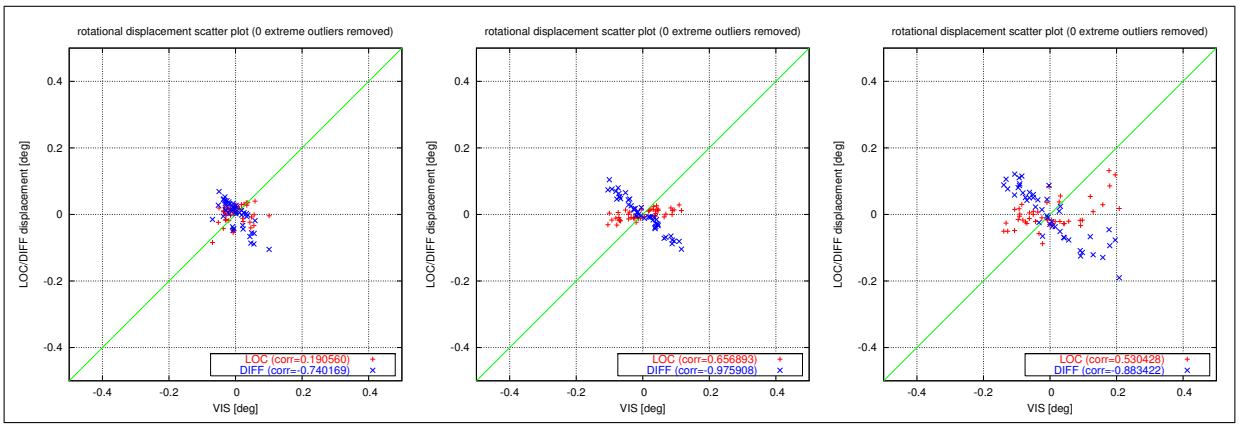


Figure 3.67: Experiment 20031217-d20: Rotational displacements for three different targets. See page 121 for an explanation of the legend.

Chapter 4

Advanced Robot Arm Control

> *"This is still just the culture shock. You wait till I've settled down into the situation and found my bearings. Then I'll start panicking!"*
>
> *Arthur Dent*

Another component of the mobile robot system is the *robot arm*, or simply *robot* in short. The word robot refers to *articulated robots* in this work, although in general public it is used much more relaxed. The first time this word was used by the czech author Karel Čapek in his 1921 drama "Rossum's Universal Robots" (R.U.R.)[1] it stood for machines which would today be called *humanoids*. Since then it has been mainly attributed to all kinds of machines that are capable of doing manipulations that are more or less vaguely similar to human manipulations, using mechanisms which are more or less vaguely similar to a human arm. Today, the term "articulated robot" is used for arm mechanisms only.

There are numerous companies building numerous types of robot arms, for example the ABB, KUKA, Stäubli and FANUC arms shown in figure 4.1. They differ in size, weight, payload, number of joints and kinematics over a large scale. Some of them are specifically designed for certain tasks, using a reduced number of joints leading to reduced kinematic capabilities, but most of them are capable of moving in all six *degrees of freedom* (DOF) of our three-dimensional world.

Figure 4.1: Some robot arms. From left to right: IRB 1400 by ABB, KR3 by KUKA, RX60 by Stäubli and ARC Mate 100iB by FANUC (images do not scale).

[1] `http://www.czech-language.cz/translations/rur-introen.html`

Being capable of performing manipulations similar to a human arm, these robot arms are mainly used in situations where manipulations would be too strenuous, dangerous or monotonous for a human being. The most commonly known application of robot arms is probably still the car manufacturing industry, although many other areas of application are common. In the car manufacturing industry the robot arms are used at assembly lines as in figure 4.2. They are used to pick and place parts that are tricky to handle, weld parts together more precisely (deterministically) than a human can do it, paint cars without the danger of damaging their health due to paint spray and many other tasks.

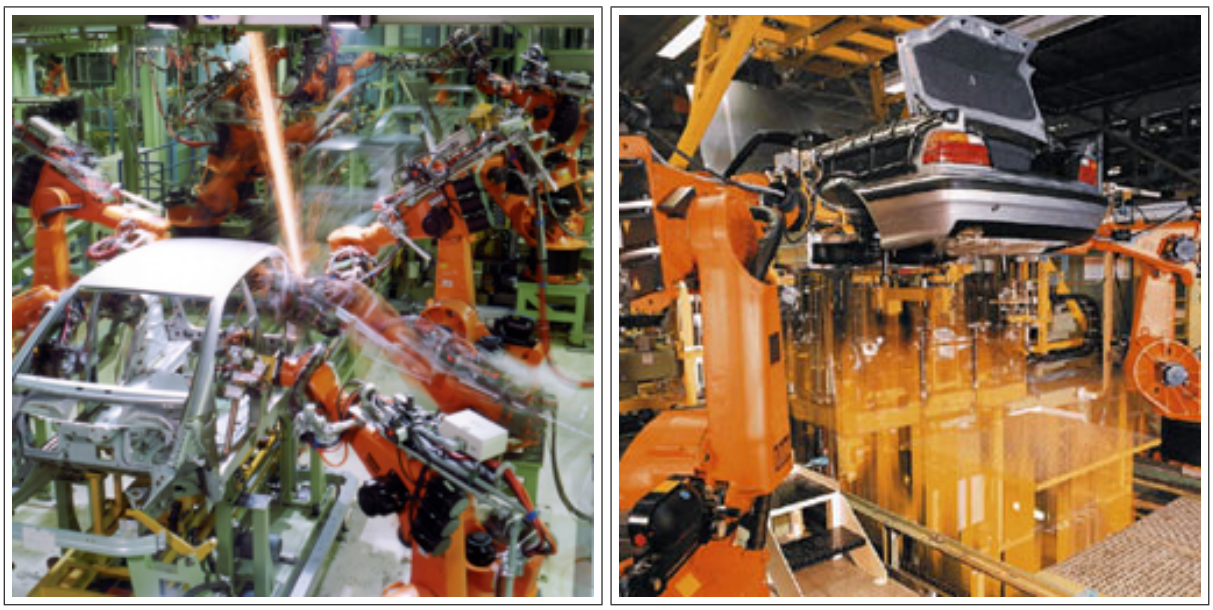


Figure 4.2: Robots at an assembly line in the car manufacturing industry.

In this work the robot arm is used to pick and place the different types of tubes, to carry them between devices and to operate some of these devices. The mobile platform “only” helps in bringing the arm close enough to the devices to allow it to perform its operations without having to group the devices close around the arm[2]. Since the the vendor specific control software of the arm has proven inappropriate for the tasks in this work a new control approach based on the well known and free scientific robot control software RCCL has been implemented.

The arm and the tool are briefly described in section 4.1. Section 4.2 introduces and compares the Mitsubishi PA-Lib with RCCL, motivating why RCCL is being used instead of the PA-Lib. The modifications and enhancements of RCCL and parts of the hardware necessary to support the PA10 are described in section 4.3. Finally, section 4.4 gives some results of the arm performance as well as a summary.

[2]See the introduction of chapter 3 for the motivation of the kinematic separation.

4.1 The Robot Arm & Tool

The robot arm used for the mobile robot system is a 1999 model Mitsubishi Heavy Industries (MHI[3], Japan) “Portable General Purpose Intelligent Arm PA10”[4]. It consists of the arm itself and a custom tool designed for the grasping tasks required for this work.

4.1.1 The “Mitsubishi PA10” Robot Arm

The PA10 arm weighs only 35 kg, yet it has a rather high payload of 10 kg. Its weight is distributed over a kinematically usable length of 950 mm, making it very compact. In comparison, for example the smallest model of Stäubli’s RX series, the RX60, weighs 42 kg at a length of 600 mm.

The arm has 7 joints arranged in a way to resemble the capabilities (not necessarily the appearance) of a human arm as in figure 4.3. It has a *redundant* kinematic, which means that there is generally an infinite number of solutions to the inverse kinematics. It also means that parts of the arm can move without changing the pose at the end effector.

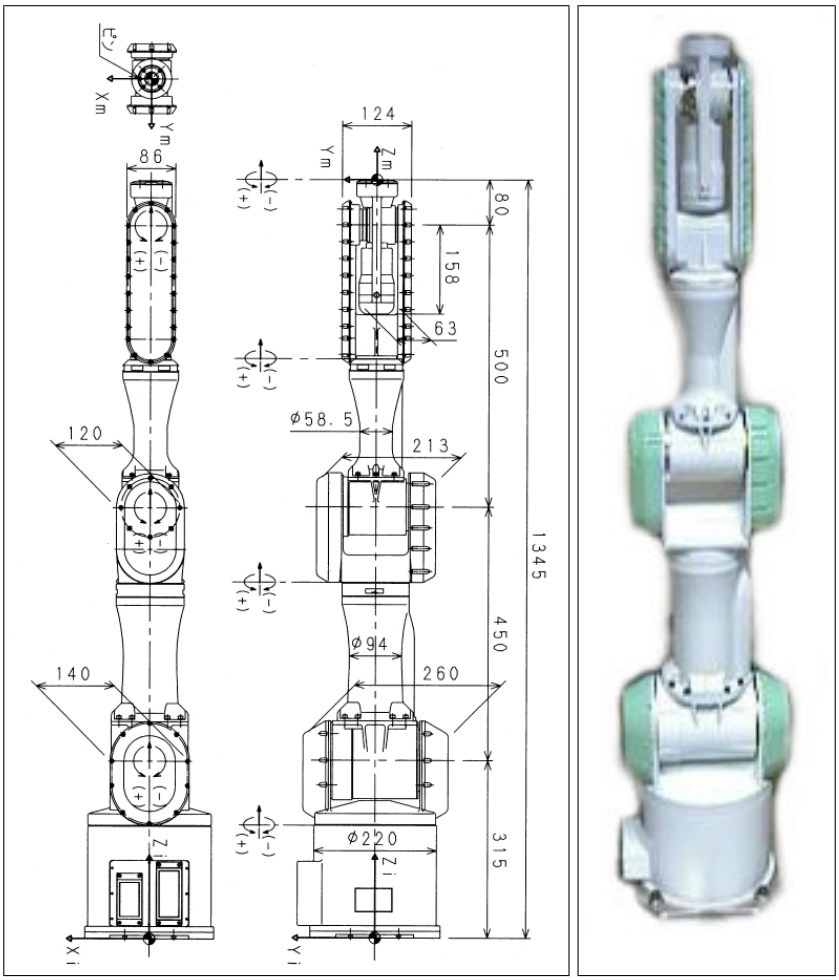


Figure 4.3: The PA10 arm with its 7 joints. All lengths are in millimetres.

[3] http://www.robot-arm.com, http://www.sdia.or.jp/mhikobe-e/products/mechatronic/e_index.html

[4] The newer PA10-6C and PA10-7C are not covered in this work, but only referenced at some places.

The arm comes with a controller box that is very compact. The logic and power boards inside it use only a fraction of the available space in the box – most of the space is used by the power supply which transforms the japanese standard 100 VAC down to 100 VDC for the power amplifiers. This small size and the fact that it is possible to run the power amplifiers with, for example, only the 48 VDC present on the mobile platform makes it possible to integrate the controller into the mobile platforms without larger modifications.

The controller implements the joint controller loops, which have to be given setpoint commands every 10 ms over an ARCNet port. The PA10 controller supports two major commands: Velocity commands and torque commands, but not position commands. This means that in order to drive any trajectory given by position points a secondary controller is needed to translate positioning errors into velocity commands.

Usually, the *motion control board* (MCB) by MHI is used as secondary controller. The MCB is an ISA or PCI board that must be plugged into the computer that is to become the controlling host for the PA10. It communicates with the controller via an ARCNet link. The MCB performs all the computations needed for moving the arm, however, no official documentation about the MCB itself exists.

The only way to use the MCB is therefore to use the "PA-Library" (PA-Lib), which comes with source code. The PA-Lib implements no functionality on its own, but only passes its function parameters to the MCB. It is therefore limited to MCB hardware features and does not allow any user enhancements.

The combination MCB/PA-Lib has some severe deficiencies which make it ill-suited for this work, most notably the path-tracking precision for arbitrary paths. These deficiencies are briefly described in section 4.2.1, where the PA-Lib is compared to the *robot control c-library* (RCCL) used in this work.

For a more verbose description of the arm see section B.1 in the appendices.

4.1.2 The Tool

The PA10 arm is equipped with a custom made tool as in figure 4.4, consisting of a microhead colour camera, a force torque sensor and an electronic gripper with specialised fingers. The purpose of the tool is to hold the different sensors and actors in order to allow the arm to grasp the tubes and manipulate the devices properly. Since it is very important not to waste major parts of the operational range of the arm, this tool has been designed to be as compact as possible.

The force sensor is a 6 *degrees of freedom* (DOF) model and used to detect forces during contact with an object, for example when inserting a tube into a slot or pressing a button. It is the only tactile sensor the arm has got, meaning that there are no explicit force sensors in the fingers or elsewhere. As such the force detection is very indirect and not suited for true tactile sensing.

The gripper is a small electric parallel jaw gripper and not, as they are in widespread use especially in industry, a pneumatic model. The disadvantage of pneumatic grippers is that they more beat their fingers on the objects rather than firmly grasp them. With small and lightweight objects this can well mean to make them slip out of the gripper, which is a severe problem if no tactile sensors exist to detect this situation. An electric gripper closes its finger much more slowly and therefore carefully. The only disadvantage of an electronic gripper is that it has a lower grasp force, but with the lightweight tubes used in this work with is not a problem.

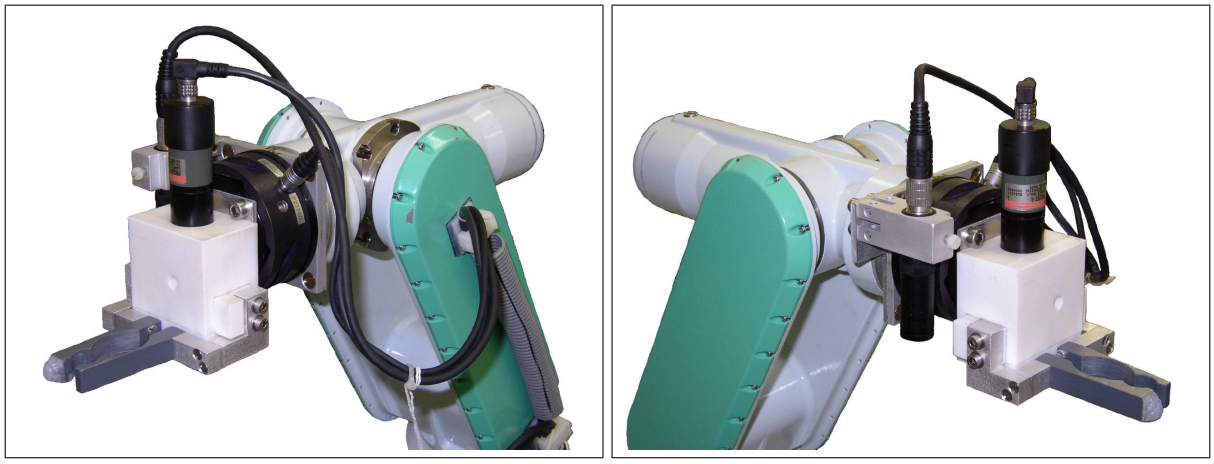


Figure 4.4: The tool with the camera, force/torque sensor and gripper.

The gripper fingers are custom made to fit to the tasks required by this work. To open lids the fingers have to reach behind a handle and lift it, which can be done with any almost finger design. To press buttons on a keyboard they are equipped with a rubber point, which can also be done with any almost finger design. To grasps tubes special care has to be taken.

The fingers have cut-outs milled into them so that they properly align with the round shape of the tubes (see figure 4.5). The reason for this is that with no cut-outs the contact to the objects would be a two-point contact only, and therefore very unstable. With cut-outs, and eventually cut-outs coated with some rubbery material the contact is established plane-on-plane, and therefore much more stable.

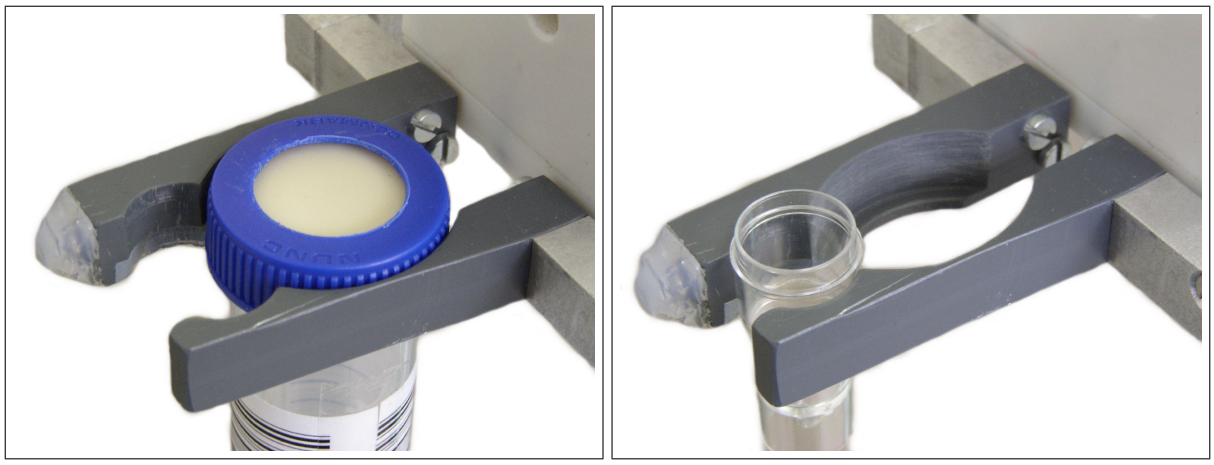


Figure 4.5: The gripper fingers holding a 50 ml NUNC tube (left) and a 1 ml Cedex tube (right).

The camera belongs to the vision system and is described in chapter 5.

For a more verbose description of the other components of the tool see section B.2 in the appendices.

4.2 Robot Control Software

Apart from the proper hardware a sophisticated *software* is also needed to actually move the arm. Motion trajectories have to be planned and split into discrete setpoints, forward and inverse kinematics have to be computed on these setpoints, command data has to be sent to the robot in real-time while several parameters of the motions like speed and acceleration have to be changeable and subsequent motions have to be blended into each other smoothly. These tasks all involve complex computations with special robot data structures and algorithms which the average industrial user does not always want to be aware of. Commercial vendors therefore ship some kind of proprietary programming language with their robots with which these users may be fully satisfied, but a scientific research project is much more demanding. This section therefore introduces the capabilities of the Mitsubishi PA-Lib in comparison to the capabilities of the free RCCL library developed by computer science researchers and being widely used at research institutes all over the world.

4.2.1 The PA-Library

As has been mentioned in section 4.1.1 MHI provides the MCB / PA-Lib package for operating the PA10. The PA-Lib is the only documented way to access the MCB and therefore the only high-level way to access the PA10. Its key features are briefly described here, explaining the reasons why it was found to be not suitable to be used for this work.

The PA-Lib originally comes as both a pre-compiled Windows library and as C source code. The source code reveals that it does not implement any functionality on its own, but only interfaces the hardware – function parameters are copied through several structs before being sent to the MCB. It does in particular not use any low-level operating system functions, which means that a port to Linux is quite trivial. As a result of this, it does not require a dedicated PC.

Compared to the V+ language and operating system that is used on Stäubli RX robots this is an advantage. The RX robots come with a proprietary robot controller on which all the robot related software has to reside. On this controller the V+ programs are *interpreted* rather than *compiled*, which means that any high-level computations that are not covered by a single V+ statement are rather slow[5]. Programs are limited to what the V+ language supports, so if the task requires anything – like advanced vision or complex and lengthy on-line calculations – that cannot be done on the controller it has to be distributed to other computers. The network communication that arises from this is in turns again limited to what the V+ language and operating system supports, with the result that it is – if at all possible – quite awkward compared to mainstream programming languages.

The PA-Lib – on the other hand – does not need an external proprietary controller for demanding computations, but can be used on any Windows/Linux computer directly. Since it can be linked to any C/C++ program it gives the user more flexibility and freedom. With the PA-Lib the user has all the abilities to

- set basic motion parameters (speed, acceleration, ...),
- do joint interpolated motions,
- do cartesian interpolated motions,

[5]For example the “execution” of an empty comment line already takes a non-neglectable amount of time.

- modify motions in real-time (very limited capability) and
- store and execute teached motions

as well as to use one of the fastest and most commonly used programming languages, likely eliminating the need to learn a new "robot programming language". If complex high-level calculations are needed they can also be done on the controlling computer without problems and without the need for network communication, though this is very easily possible if required. This is aided by the fact that *any* computer may be used, and if one should be found to be too slow replacing it by a more powerful one is a cheap and easy thing.

These features are, however, not enough for a scientific user. One key interest of a scientific user it to have the *possibility* to get control over all high and low-level functions and to add new functionality if the provided one is not sufficient. This, however, is explicitly not possible with the PA-Lib.

With the PA-Lib the user has a certain number of routines implementing a certain functionality. All this functionality and its parameterisation is limited to what MHI considers important. For example, the support for modifying some motion parameters at real-time does not allow a full scale force control because it is only blends very softly over the existing motion and is therefore too sluggish (see section 4.4.2). If the user is at any point not satisfied with the performance of the PA-Lib there is absolutely no way to add better routines because the PA-Lib does not offer any support for enhancing its functionality and no documentation is available that might indicate whether this is at all possible with the underlying MCB. One particular thing that is completely impossible are user-level callback functions that might be used to add low-level behaviour that is otherwise not achievable.

Summarised, the major shortcomings of the PA-Lib are:

- It does not supply any mathematical routines,
- the real-time modifications of motions perform too badly to be used for force control and
- it offers no support to add low-level functionality to overcome its own shortcomings.

Because of these shortcomings the PA-Lib has been found unsuitable for the requirements of this work and has – apart from some initial tests – not been used. Instead the freely available and well-known robot control software RCCL has been chosen to be adapted to the PA10.

4.2.2 The "Robot Control C-Library" RCCL

RCCL is the "Robot Control C-Library" by John E. Lloyd and Vincent Hayward[6]. Its first versions date back to the mid-80s, but it is still technologically up-to-date as far as low-level robot control is concerned. Its objective is to give the user access to different types of robot hardware, the kinematic functions, mathematical structures as well as a lot of routines to do daily work, including a trajectory generator for the actual motions. Its particular strong point is the ability to install user functions in the trajectory generator that can modify all kinds of data structures and thus freely determine the path of the motion. It is not limited to motion types that have been designed in advance but allows the user to add whatever might be needed later on.

[6] See [Lloyd and Hayward 1989].

RCCL was the first robot library written in C to run on a normal UNIX workstation. Compared to the proprietary programming languages running on proprietary controllers this is a great advantage in several aspects. One aspect is that the user does not have to learn a – however trivial – new programming language for each new type of robot. Another aspect is that auxiliary mathematical routines are either available or can at least be trivially added. But the really important aspect is that the robot can be controlled from the same computer and even the same program that is used for computing the targets in the first place – communication with other computers is only seldomly really required.

Commercial robot control software like the V+ language is often too limited to implement expensive high-level computations in. These languages are basically for executing motion commands only. Anything that requires more computing power is likely to require to distribute the task to a more powerful machine, which in turns requires network communication which can be quite awkward with these systems. With all software implemented in a single programming language that offers enough functionality and integrated on a single computer that is cheap and has enough power this problem does not occur and the user can instead concentrate on the real task.

The PA10 does a first step in improving this situation because the PA-Lib is written in C and the MCB can be plugged into any computer, however, the PA-Lib / MCB combination is too limited to take full advantage of this first step. It lacks a lot of useful functionality, and – since it is limited to the undocumented MCB hardware – most notably the possibility to add it later on. RCCL is very well documented down to the hardware interfaces and has been used by the author of this work for several years. Its source is available and allows an even further insight into all details than the documentation. The only drawback – if any – is that it is no longer supported and that it may not be used commercially.

Section B.3 in the appendices gives an optional introduction into some key features of RCCL listed here. None of these features has a counterpart in the PA-Library, but some of these features are absolutely mandatory for this work. The features are:

Mathematic Computations : A very trivial but very disturbing – if lacking – point are the mathematical helpers around the robot data types. They include functions to do computations with sets of joints, vectors, transforms – 4×4 homogenous matrices describing relative spatial coordinate changes – and kinematics. These mathematical helpers are very important in the every day work because without them parts of the robot's physical functionality cannot be used. For example, the PA-Lib does have a function to move a robot to a target given by a transform, but does not have any routine to do calculations with transforms. As a result the function is pretty useless.

Position Equations : The most important mathematical construct is the position equation, where the target of a motion is specified by two separate chains of transforms. One of these chains must contain the robot transform. The trajectory generator takes a position equation, solves it for the target robot transform, compares this with the current robot transform and issues a motion to eliminate the difference.

Using position equations the target description can be split into separate components, so that if a rack with 10 tubes is moved on the table only the relative position (transform) of the rack on the table has to be changed and not all 10 positions of the tubes. The PA-Lib does not have such a construct.

Arbitrary Motions : Any robot control software has a fixed number of motion types built-in, usually joint and cartesian interpolated straight line motions and circular arcs. RCCL allows the user to attach *control functions* to transforms to modify them in real-time and thereby change the motion path. This construct can be used to implement completely new motion types as well as force control (which is nothing more than a local modification of a motion). The PA-Lib does not have such a construct.

Singularities : One application of arbitrary motions are *singularities* – positions where the robot can for kinematical reasons not follow all motions as issued by the trajectory generator because it would exceed the joint velocity limits. If such a situation cannot be avoided by other means it has to be dealt with at motion generation level, but for this to be possible there must be a way to change motions. With RCCL this can be done by means of control functions, but the PA-Lib does not have such a construct.

(Virtual) Robot Cooperation : In RCCL several robots can cooperate if their position equations contain the other robots transform. These robots need not be real, but can also be virtual. A virtual robot (object) does not have joints, but only its transform. This transform can be contained in the position equation of a real robot which is then asked to endlessly track its current position. As a result it will follow the motions of the object without the user having to do any calculations. The PA-Lib does not have such a construct.

4.2.3 Consequences

It is mainly due to these superior capabilities and flexibilities of RCCL that using only the PA-Lib has never been considered an option. The initial approach instead targeted at a mixture of RCCL and the MCB/PA-Lib where RCCL should be used to generate joint position setpoints which would then be passed to the MCB operating in “each axes real-time mode” (`pa_mod_axs()`) with the special `pa_odr_axs()` PA-Lib function. Not anticipating the fact that this approach proved to be too badly conditioned to be usable and so finally a *full* PA10 support had to be added to RCCL even the initial approach requires modifications of RCCL which shall now be discussed.

4.3 RCCL on the PA10

RCCL originally only supported the PUMA, SCARA and other even lesser known old type robots, basically because no noteworthy development work to add new robots has gone into it for a couple of years, in turns basically because even its inventor considered it too old, cumbersome and scheduled to be replaced by a successor. The reasons and consequences still apply today, but otherwise the situation has not changed.

As a result support for the PA10 as required by this work has been implemented. Communication to the arm controller over an ARCNet network has been established, software joint controllers to translate from RCCL’s position commands into the PA10’s velocity commands has been integrated and the PA10 kinematic has been implemented. The changes are described in detail in the following subsections.

4.3.1 Kinematic

The PA10 with 7 joints is a redundant robot and as such has a kinematic that is incompatible with any robot supported by RCCL so far. In particular none of the officially supported robots had more than 6 joints, so a lot of structures and functions were limited to that. Before implementing the PA10 kinematic some data structures and functions therefore had to be modified.

Basically, RCCL used/uses the constant "`#define MAXJNTS 12`" as defined in `include/maxjnts.h` as a limit, but only to size a couple of secondary arrays. The primary array defining the set of joints on an actual robot is sized separately. This original joint specification is given as

```
#define NUM_JNTS 6

typedef struct {
  float v[NUM_JNTS];       /* joint values */
  char type[NUM_JNTS+1];   /* joint type ('r' or 'p') */
  int num;                 /* number of actual joints */
} JNTS, *JNTS_PTR;
```

in `include/rcclDefs.h` and does have a `num` field indicating how many joints the robot really uses. Incrementing `NUM_JNTS` from 6 to 7 only revealed that a lot of functions had `6` hardcoded as limit rather than `MANIP->JNTS->num`, so the general porting efforts had to go beyond increasing `NUM_JNTS`. The kinematic functions themselves could only be established after these changes.

Kinematic (greek: *kinema* = "movement") is the science of movement and covers both the *dynamic* aspects of motion as well as the *static* aspects of position, because motion is nothing more than a change of position. In robotics, however, the term "kinematic" is mainly used to refer to positional questions only, not dynamical ones[7]. Kinematic in this context means to compute cartesian robot positions from joint values, and vice versa. These computations need a description of the physical appearance of the robot like the Denavit-Hartenberg (DH) convention[8]. Any such description yields a set of parameters like in table 4.1 describing each segment individually, which are combined to describe chains of segments up to the complete arm. RCCL does not use a generic kinematic routines that utilises a full DH parameter set, but individual functions for each class of robots instead. The reason for this is simply that RCCL is rather old and had to run on computers that would nowadays considered to be mere pocket calculators, so it had to take advantage of all possible simplifications. For the PUMA robot class for example a lot of DH parameters are either zero or at least equal for all PUMAs, so the corresponding simplifications are hardcoded in the appropriate routines. As a result of this adding a new kinematic to RCCL not only means to add a new set of DH parameters, but actually to implement new routines.

The forward kinematic then is the problem of computing a cartesian position from a set of joint angles. This task is trivial and could – if needed – be quite easily implemented in a generic way for n-joint robots: Each segment j can be described by a single transform T_j that is defined by the DH parameters (including the joint variable). Multiplying together these (7, in case of

[7] The question of robot dynamics does of course exist, but is explicitly named "robot dynamics" and not "robot kinematics".

[8] See [Denavit and Hartenberg 1955] and [Denavit and Hartenberg 1964].

i	θ_i	d_i	a_i	α_i
1	π	0	0	$\frac{1}{2}\pi$
2	π	0	a_2	0
3	0	d_3	0	$\frac{1}{2}\pi$
4	π	d_4	0	$\frac{1}{2}\pi$
5	π	0	0	$\frac{1}{2}\pi$
6	0	0	0	0

Table 4.1: Denavit-Hartenberg parameters for a PUMA-260 under RCCL.

the PA10) T_j yields the end effector position[9]

$$T_{\mathrm{ROBOT}} \;=\; T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 \cdot T_6 \cdot T_7$$

The PA10 forward kinematic grants itself the same simplifications as the existing PUMA routines and combines these multiplication in a single complex but computationally cheaper expression. The forward kinematic is unique, meaning that each single input always produces exactly one single output (although several different inputs may produce the same output). It needs no further attention.

The inverse kinematic is the inverse problem of computing a set of joint angles from a cartesian position. This computation needs the additional attention that it is not unique, meaning that a single input can produce multiple outputs. In case of the PUMA kinematic it means that there are 8 possible sets of joint angles for each legal cartesian position (which do – as in case of singularities – not always have to be really different). In order to chose one of these solutions RCCL provides a bitmask indicating the desired configuration.

In case of the PA10 kinematic the situation is even more problematic because a closed form for the inverse kinematic does not exist for redundant robots. Redundant robots do not only have a countable multitude of solutions but rather an infinite number, which means that providing a bitmask to select a solution is not sufficient. There are ways of controlling redundant robots but they use other methods than the direct inverse kinematic, for example the pseudo-inverse of the jacobian. Since RCCL does require a direct inverse kinematic a compromise has been established.

Instead of using all 7 joints the implemented kinematic only uses 6 joints, keeping the redundant one (J_3, called "E1" by MHI) constant at zero[10]. The resulting "reduced" PA10 kinematic shares an interesting feature with the PUMA robots, which is that the rotation axes of the last three joints intersect in one point. These robots are called *wrist partitioned* robots and allow a computational split of the kinematic: As can be seen from the upper right 3×3 matrix of zeros in the jacobian for a PUMA-260 in

[9]The reason why this T_{ROBOT} is also called T_6 is historical, motivated by a time where no robot had more than 6 joints. In this work it is mostly called T_6 because the individual T_j for joint number 6 is not of interest here.

[10]This kinematic has been developed at a time where only the original PA10 with 7 joints existed. Today the newer PA10-6C and PA10-7C versions are 20 mm shorter in the last segment than the PA10, but since except from this the placement of the joints is exactly the same and the joint that is missing at the PA10-6C is exactly the redundant joint the same kinematic routines can be applied to them.

$$J = \begin{bmatrix} -s_1 r_1 - c_1 d_3 & c_1 r_2 & c_1 c_{23} d_4 & 0 & 0 & 0 \\ c_1 r_1 - s_1 d_3 & s_1 r_2 & s_1 c_{23} d_4 & 0 & 0 & 0 \\ 0 & -s_{23} d_4 - c_2 a_2 & -s_{23} d_4 & 0 & 0 & 0 \\ 0 & -s_1 & -s_1 & c_1 s_{23} & -s_1 c_4 - c_1 s_4 c_{23} & -s_1 s_5 s_4 + c_1 r_3 \\ 0 & c_1 & c_1 & s_1 s_{23} & c_1 c_4 - s_1 s_4 c_{23} & c_1 s_4 s_5 + s_1 r_3 \\ 1 & 0 & 0 & c_{23} & s_4 s_{23} & c_5 c_{23} - c_4 s_5 s_{23} \end{bmatrix}$$

where $s_i = \sin(\theta_i)$, $s_{ij} = \sin(\theta_i + \theta_j)$, c_i and c_{ij} respectively and

$$\begin{aligned} r_1 &= s_{23} d_4 + c_2 a_2 \\ r_2 &= c_{23} d_4 - s_2 a_2 \\ r_3 &= c_5 s_{23} + c_4 s_5 c_{23} \end{aligned}$$

as abbreviations, the last the joints do not contribute to the *position*. Only the first three joints 1 to 3 determine the position and the last three joints 4 to 6 only correct the *orientation* to the desired value[11]. This does not present a noteworthy gain for the forward kinematic, but for the inverse kinematics it allows to operate with much shorter expressions than with a generic 6-joint kinematic. It is only because of the fact that the RCCL routines for the PUMA kinematic have special assumptions about some of the DH parameters hardcoded that they cannot be used for the PA10.

Because of this simplification the implemented kinematic is "only" equally powerful than the PUMA class kinematic, in particular it is not possible to use the redundancy in cartesian motion. This is not a noteworthy restriction as the task requirements do not explicitly require a redundant robot in the first place, and should it yet become necessary it can be added later by means of arbitrary motions in RCCL. Using this kinematic the PA10 can be fully used in joint interpolated mode (which only uses the forward kinematic) and in a limited way in cartesian mode (which also uses the inverse kinematic). The only thing that has to be externally ensured is that the redundant joint J_3 really *is* zero when a cartesian motion is started because otherwise it would jump to zero in the first cycle(s), which could damage both the robot and anything in its vicinity.

The PA10 jacobian has not been established because it is not needed for this work. Given the forward kinematic its computation is trivial.

4.3.2 ARCNet Interface and Protocol

Another area of modifications involves the ARCNet communication with the PA10 controller. ARCNet (*Attached Resource Computer NETwork*) is a network communication hardware and protocol like ethernet, but different in some aspects. Electrically it defines – like ethernet – several transmission channels like coaxial or twisted pair copper or fibreglass cables using bus or star topologies and several modulations techniques for transmitting data. Logically it defines a token-ring protocol that ensures that no collisions can occur - one node (the one with the highest id) in the network is the master and controls the traffic. It is probably because of the

[11] See [Scherer 1998], chapter 4.2.

token-ring characteristic of being deterministic and real-time capable that MHI chose ARCNet as communication media[12].

MHI only provides the MCB to access the PA10 controller, which means that any user who does not want to use it has to use an ARCNet board. Since ARCNet is a niche product there are only few companies building ARCNet boards, and so the choice is limited. Unfortunately this is a problem because at the same time MHI uses media transceivers which are rather uncommon in Europe. This applies to both the original PA10 used in this work as well as the newer PA10-6C/PATENSEVEN. Their ARCNet hardware specification is given in table 4.2.

	PA10	PA10-6C/PA10-7C	optional
chip	SMSC COM20020	SMSC COM20022	
speed	5 Mbps	10 Mbps	
transceiver	TMC HYC2485S	Agilent HFBR-1528/2528	TMC HYC4000
	AC coupled	DC coupled	AC coupled
	“RS485 type”	plastic optical fibre (POF)	“RS485 type”
media	twisted pair	1 mm POF	
		650 nm wavelength (red)	
connector	RJ11	Agilent “Versatile Link”	

Table 4.2: The ARCNet hardware specifications of the PA10, PA10-6C and PA10-7C. The term “RS485 type” is written in quotation marks because it comes from TMC/MHI, but the mode is actually incompatible to what is known as *true* RS485 mode (see section 4.3.2.1).

4.3.2.1 PA10 ARCNet Hardware

The original PA10 used in this work uses a HYC2485S transceiver by TMC[13] to transmit data with 5 Mbps over twisted pair cable with RJ11 connectors. TMC and MHI call this a “RS485 type” transmission, though it is in fact incompatible to what is known as *true* RS485 mode. RS485 operates with (differential) DC voltage signals, whereas the HYC2485S/HYC4000 use an AC modulation. As a result of this the RS485 ARCNet boards by companies like SOHARD[14] or Contemporary Controls[15] can not be directly used to interface the PA10 controller.

Instead, a SH-ARC-PCI board from SOHARD has been modified by replacing its transceiver with a HYC2485S. Only using this modified board the PA10 can be accessed from a computer with virtually any operating system because the SMSC[16] COM20020/22 ARCNet chip family is actually the only ARCNet chip on the market and every driver supports it.

[12] In case of just a peer-to-peer “network” with only two nodes and only one application communicating over it ethernet could probably also be used, but that discussion is not of interest for this work.

[13] Toyo Microsystems Corporation (now part of SMSC). `http://www.smsc.jp/en` and `http://www.arcnet.com/toyo.htm`

[14] SOHARD AG, Würzburger Straße 197, D-90766 Fürth, Germany. `http://www.sohard.de`

[15] Contemporary Controls Ltd., Barclays Venture Centre, University of Warwick Science Park, Sir William Lyons Road, Coventry, CV4 7EZ, England. `http://www.ccontrols.com`

[16] Standard Microsystems Corporation (SMSC), 80 Arkay Drive, Hauppauge, NY 11788-8847, USA. `http://www.smsc.com/main/catalog/arcnet.html`

4.3.2.2 PA10 Protocol

To speak to the PA10 controller MHI uses a protocol that defines several commands. Since the MCB cannot be used by 3rd-party software this protocol is the only way to access the controller. As documented in [Mitsubishi PA10 Servo Driver Manual], the commands implement a simple state machine as in figure 4.6.

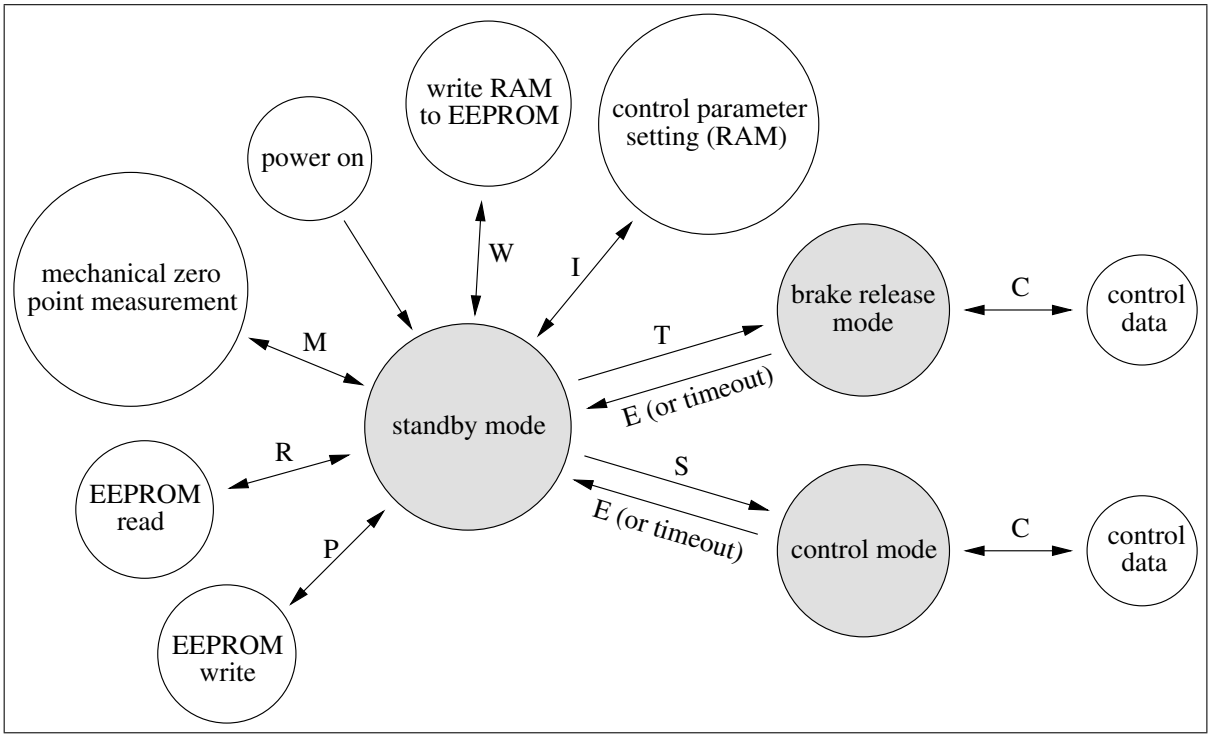


Figure 4.6: The state machine implemented by the PA10-6C controller protocol.

Apart from several diagnostic and maintenance commands the primary commands are the "S" and "E" commands to bring the arm in or out of "control mode" and the "C" command. The specification of these commands is as follows:

- **The "S" command** is used for *starting* arm control. After power-up the controller is in "standby" mode, waiting to be initialised. Only after receiving a "S" command it enters "control" mode where the arm can be used. Additionally it starts a watchdog timer to wait for "C" commands. Any sending of "E" commands, watchdog timeouts or other errors result in the controller going back into the standby mode.

- **The "C" command** is used for *controlling* the arm. When the controller is in "control" mode this command can be used to release/activate the brakes, to enable/disable the individual servo drivers[17], to send them velocity or torque commands and to read the current position During normal operation the controller expects a "C" command every cycle of 10 ms (on the PA10, or 2 ms on the PA10-6C/PA10-7C) and defines a timeout as a certain multiple of that time. If the higher level fails to send the next "C" command within

[17]Note that with the "S" command the servo driver of an axis *has* to be enabled when its brake is released. Only if the "T" command has been used to enter (a similar) control mode the servo driver may remain deactivated, thus allowing to passively move the arm in a zero-gravity way.

that timeout the controller goes back into standby mode, stopping the arm and activating the brakes.

- **The "E" command** is used for *ending* arm control. When arm control is ended the servo drivers are disabled, the brakes are activated and the controller goes into standby mode. Only after sending a new "S" command the arm can then be used again.

This protocol has some drawbacks that affect any kind of robot control software. One drawback is that multi-byte data inside the packets is encoded in little endian (Intel) order[18] on the original PA10 used in this work and in big endian (Motorola) order on the newer PA10-6C/PA10-7C. In addition to this the parameters in at least the "C" command differ slightly. As a result the PA10 and PA10-6C/PA10-7C protocols are incompatible and a unified driver for both types of robots has to be aware of this.

Another drawback is derived from the fact that all these commands strictly follow a client/server principle. It is only the host computer that actively issues commands and only then the controller passively sends replies. It does in particular not report the current joint values (or other state information) on a periodical base, meaning that the host computer has to actively ask for them. Since there is no separate command for requesting joint values but only the "C" for sending velocity commands that returns the joint values as a reply this makes it troublesome to obtain current joint values, as for example to be used in the PID controller described in section 4.3.3 and used in this work to generate the velocity setpoints in the first place.

4.3.3 Joint Controllers

RCCL has a trajectory generator that works by planning a true desired trajectory from the start to the target, computing cartesian setpoints along that trajectory and directly applying the inverse kinematic on these cartesian setpoints to obtain joint setpoints. As such it explicitly produces joint *position* setpoints to be passed to the robot as in figure 4.7. The reason it does this is that the PUMA – like all robots RCCL supported so far – accepts position setpoints only.

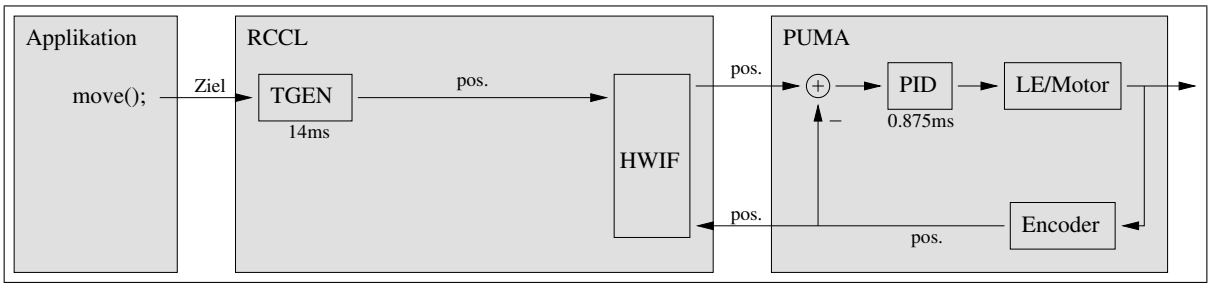


Figure 4.7: RCCL's interaction with the PUMA controller: Position setpoints are sent every 14 ms. The PUMA controller then has 16 of its internal cycles time to reach the target.

With the PA10 this does not work any longer because the PA10 only accepts *velocity* setpoints. The only way to produce velocity setpoints from position setpoints without rewriting the TGEN is to add an additional PID controller to RCCL that translates between the values as in figure 4.8.

[18] Big endian byte order means that the 4-byte word 0x12345678 becomes a byte stream of 0x12 0x34 0x56 0x78 and little endian byte order means that it becomes a byte stream of 0x78 0x56 0x34 0x12.

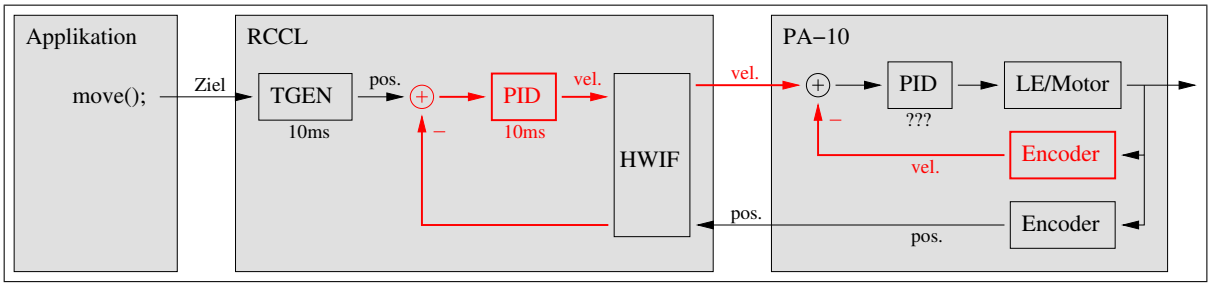


Figure 4.8: RCCL's interaction with the PA10 controller: In order to be able to send velocity setpoints RCCL must use an internal PID controller. Since this controller can not run faster than the PA10 controller cycle rate and should not run slower than RCCL's TGEN this means that the PID controller has only 1 cycle time to reach its target.

This approach does work, but introduces a few problem. First of all, a general principle of control theory states that when using cascaded controllers the inner ones have to run several times faster than the outer ones. The reason is simply that the inner controller must be given some time to react to changes of the outer controller. This scheme is, however, not possible in case of three cascaded controllers here because the cycle rates of the outer- and innermost one are fixed:

- Since the PA10 servo drivers only accept new setpoints every 10 ms there is no point in letting the PID controller compute its output faster, so it is bound to not run faster than every 10 ms. Actually, the servo drivers should be given some time to react, so the requirement can be given as $t_c \gg 10\ ms$.
- Since RCCL's TGEN produces new outputs every 10 ms there is not point in letting the PID controller accept its input slower, so it is bound to not run slower than every 10 ms. Actually the PID controller should be given some time to react, so the requirement can be given as $t_c \ll 10\ ms$.

The only way to obtain a halfway legal combination is to reduce these requirements from $t_c \gg 10\ ms$ and $t_c \ll 10\ ms$ to $t_c \geq 10\ ms$ and $t_c \leq 10\ ms$ and let the PID controller run at *exactly* 10 ms too, but this introduces the announced problems: The PID controller now has only 1 cycle time to reach its target before a new target comes in, which can be proven to be impossible to do. As a result it will *never* reach its target in the desired time but *always* lag more or less considerably behind. This is not a fully satisfying behaviour, but about the best that can be done with the given hardware.

It is of course possible to let RCCL's TGEN run slower so that the PID controller has more time to do its work, but this would only mean that the trajectory is sampled much more coarsely. Any quick movements or reactions as are required in force control would become impossible. Overall, the resulting performance can be expected to be even worse than with the implemented approach.

Another problem is that any controller needs current values for its calculations, and the PA10 does not properly provide them. As has been mentioned there is no command on the PA10 to request current joint positions, but only a command to set new targets that gives the joint positions as return values. This means that the current joint values become available only *after* the controller loop has run and computed its velocity command. This situation has to be considered

a consequence of the above problem, because if the PID controller had more cycles time to reach its target the problem would probably not show up in the first place. With just one cycle time, however, it does show up quite clearly. Several approaches are conceivable to deal with it:

1. Issue an additional "C" command at the *beginning* of the control circle – repeating the old velocity commands – with the only purpose of obtaining current joint positions, then run the control loop and immediately issue another "C" command with the new velocity values. This is technically possible, but MHI does not specify how the servo driver will react to this and therefore it cannot be considered stable and cannot be used.

2. Do nothing, but simply use the position readings from the last cycle when computing the new velocity. This basically works, but introduces 1 cycle delay in the position readings. As a result the controller can by definition never fully reach its target, the positioning performance will become overly bad and therefore the approach cannot be used.

3. Try to forecast the current position based on the values from the last cycle. If the trajectory is not changed too abruptly this should at least yield a better performance than to just use the position from the last cycle.

For this work the third approach has been implemented with a simple predictor that adds a fraction of the last velocity command to the last position reading to estimate the current position. This is still a suboptimal solution compared to a PUMA system, but again about the best that can be done with the given hardware.

Because of these two problems the resulting positioning performance can not be expected to be as good as with an old PUMA system, but still better than with the MCB and certainly good enough to allow all required operations in the biotechnological laboratory, as is shown in section 4.4.

As a final note it has to be mentioned that the newer PA10-6C/PA10-7C have a cycle time of only 2 ms compared to the 10 ms on the PA10 used in this work. Such a reduced cycle time would theoretically allow to run the PID controller 5 times faster than RCCL's TGEN without changing that one's cycle time, though due to the internal thread structure of RCCL this could be quite complicated. Whether this really improves the positioning performance has to be left an open question because it has neither been part of this work, nor had the necessary resources for the tests been available.

4.4 Results

As has been shown the usage of RCCL involves several issues that affect the performance of the arm. These issues all arise from the fundamental fact that since the PA10 uses velocity commands for the servo motors it can strictly speaking not be legally used by RCCL. Adding a joint controller to the TGEN principally allows to use it, but introduces technical complications. Control theory demands that if an additional controller is inserted into a system either the innermost controllers must run faster or the outermost controllers must run slower, neither of which is possible or admissible in case of the PA10. The performance of the resulting cascaded system can therefore never be as good as of a system which accepts position commands in the first place. This may lead to questioning whether the idea to use RCCL is in fact a good one or not, so in this section the achievable performance has to be shown.

Two different types of experiments have been done, one dealing with path tracking performance and one dealing with force control. It will become obvious that in both cases the performance of RCCL on the PA10 is in fact suboptimal, but still better than with the original MHI software. A summary will finally point out why this performance is fully sufficient for the requirements of the biotechnological service robot.

4.4.1 Path Tracking Performance

The first experiments compare the path tracking performance of the PA10 operated by RCCL (PA10/RCCL) and by the MCB (PA10/MCB)[19] with a PUMA260 operated by RCCL (PUMA260/RCCL). Two different types of motions have been done: A cartesian spiral motion in 3d-space and a cartesian square motion in 2d-space. Due to the different size of the PA10 and PUMA the paths are not 100% identical, but still comparable. A number of graphs are used to show certain aspects of the performance:

- The *commanded* and *observed* positions of joint #1 are given as an example to show the performance of the robot in following the path in joint space. This includes not only the question of whether it follows the shape of the motion at all rather than distorting it, but also the question of the time-delay with which it follows it.
- 3d and 2d-projections of the motion are given to show the performance of the robot in following the path in cartesian space. These graphs do show the shape of the motion quite nicely, but do not directly show whether the robot followed it with a time-delay or not. They do show this indirectly, though.
- Finally, the euclidian norm of the cartesian positioning error is shown. This value shows the full influence of the time-delay with which the robot follows a path because it may be non-zero even if according the 3d or 2d plots the motion matches the desired path appearingly perfectly.

4.4.1.1 Cartesian spiral Motion

In the first motion the robots are requested to move along a cartesian spiral which lies tilted in the 3d-workspace. This motion is very harmonic in cartesian space and therefore produces very harmonic, sine-like motions in joint space. The absence of high accelerations presents a rather optimal case for the joint controllers and should allow them to follow the path just perfectly. On the other hand a spiral is a long motion that keeps changing its direction, so if a robot should be lagging behind in time this is likely to show up as cartesian deviation off the trajectory.

Figure 4.9 shows the cartesian results from this experiment. In the 3d and 2d-plots of the cartesian position during the motion it can be seen that the PA10/MCB lags so much behind that it completely leaves the desired trajectory. It takes “shortcuts” through the open space to reach a setpoint that is already far ahead along the spiral, but never actually reaches that point. The PA10/RCCL performs much better in this aspect, though small deviations are still visible. Only for the PUMA260/RCCL no deviations can be seen.

[19] More precisely, RCCL is also used in this case to generate the setpoints because the MCB cannot generate spiral motions, but the setpoints are then passed to the MCB. This is done by using a special version of RCCL that uses the PA-Lib as hardware interface. Because of the results of these experiments this version has never been used again.

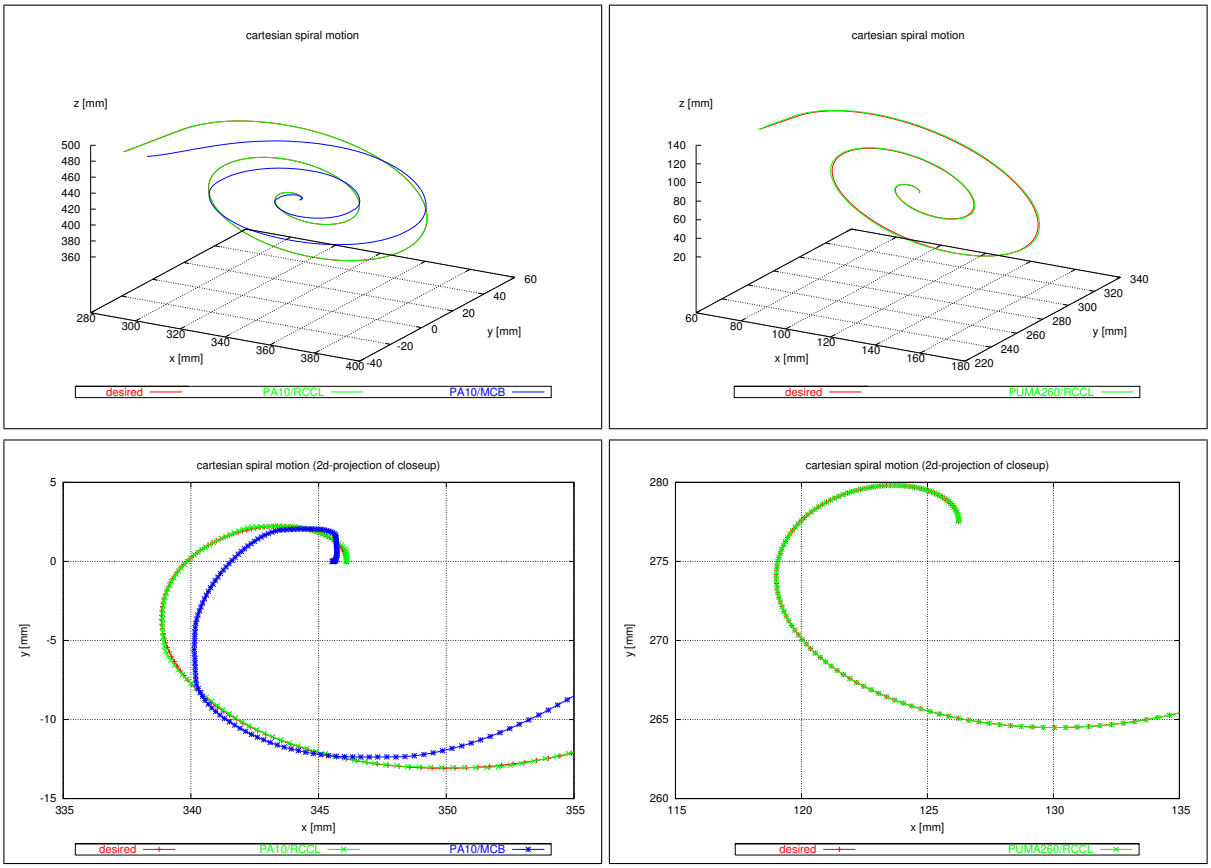


Figure 4.9: Cartesian results from the cartesian spiral motion. The first row shows the original motion in 3d and the second row shows a 2d projection of the centre of the spiral. The left column shows results from the MCB and RCCL control for a PA-10 and the right column shows results from the RCCL control for a PUMA-260 for comparison.

This cartesian deviation is caused by a sluggishness at joint level, which is shown in figure 4.10. It can be seen that the PA10/RCCL lags slightly more behind than the PUMA260/RCCL, but by far not as much as the PA10/MCB. Due to the extreme lag the PA10/MCB builds up a *mean square error* (MSE) of 1.14 degrees with a maximum peak of 3.79 degrees where the PA10/RCCL only has 0.22 degrees with a maximum peak of 0.77 degrees. This means that the RCCL solution performs about a factor of 5 better than the MCB solution. The PUMA260/RCCL has only 0.13 degrees with a maximum peak of 0.34 degrees.

Figure 4.11 shows the euclidian norm of the cartesian displacement between the desired and observed position. Since this measure includes both deviations along and off the trajectory it is generally larger than the 3d and 2d-plots might suggest. As can be seen the PA10/MCB has a MSE of 18.3 mm with a maximum peak of 46 mm where the PA10/RCCL only has a MSE of 2.4 mm with a maximum peak of 4.9 mm. The RCCL solutions performs better by a factor of 7-8. The PUMA260/RCCL with 1.3 mm with a maximum peak of 2.5 mm performs again better than the PA10/RCCL by a factor of about 2.

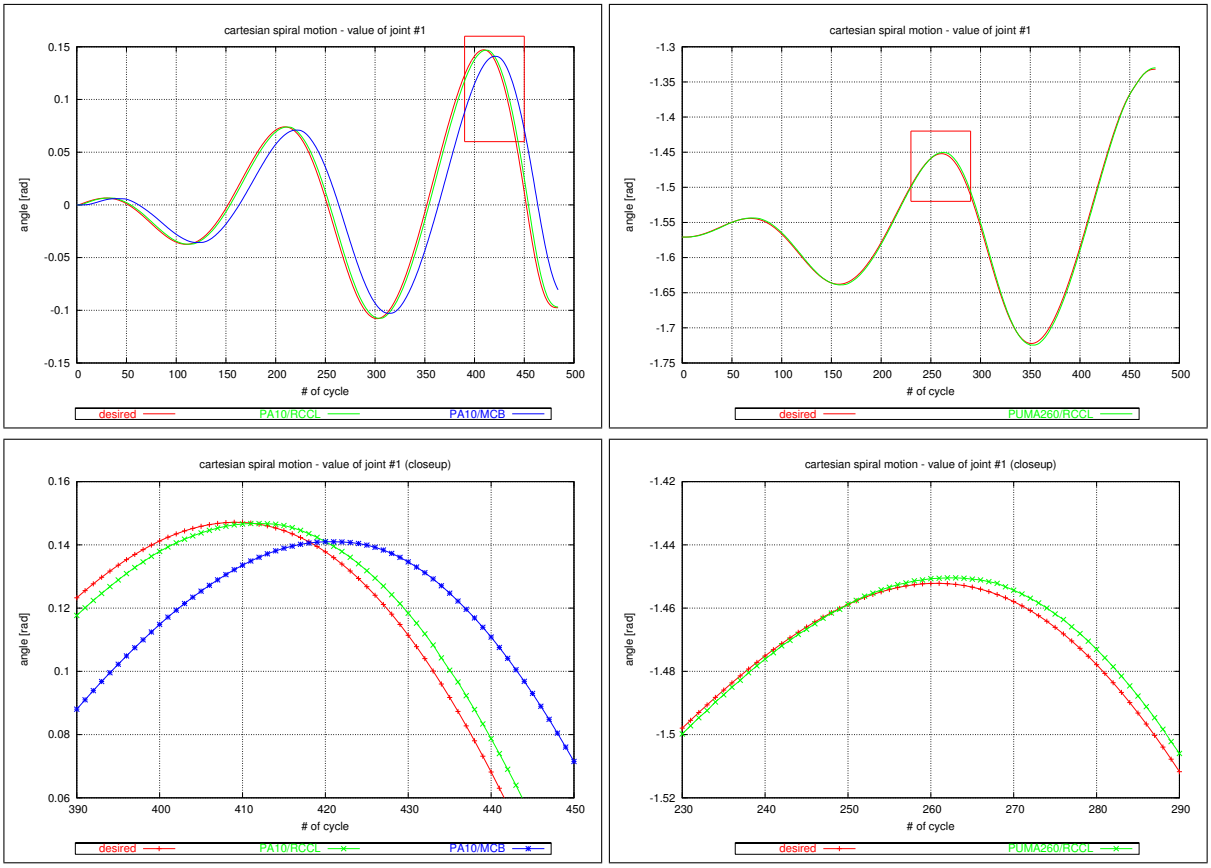


Figure 4.10: Joint results from the cartesian spiral motion. The first row shows the motion of joint #1 and the second row shows a closeup of the area marked in the first row. The left column shows results from the MCB and RCCL control for a PA-10 and the right column shows results from the RCCL control for a PUMA-260 for comparison.

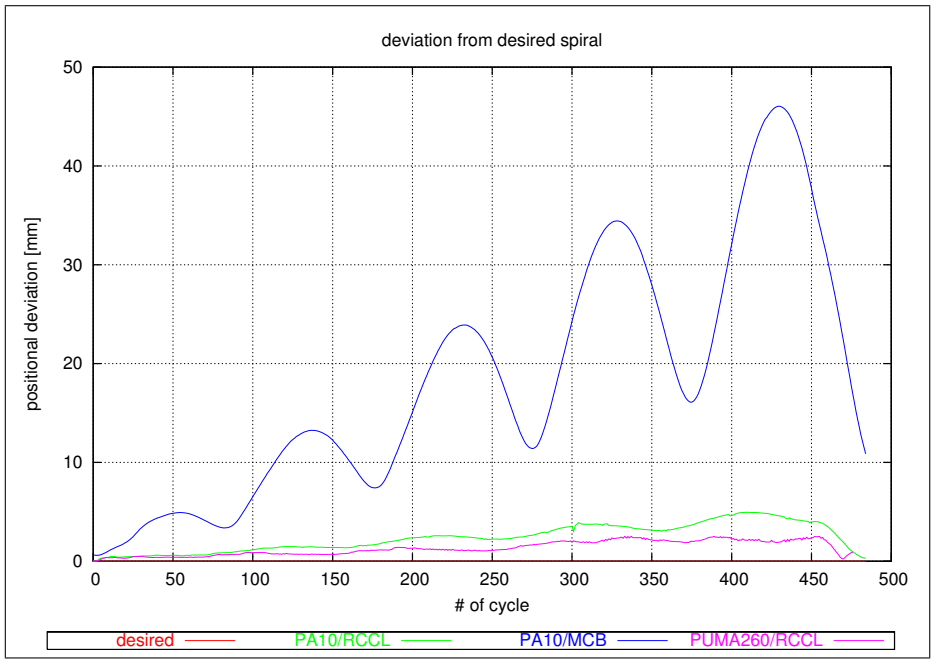


Figure 4.11: Euclidian norm of the cartesian positioning error during an spiral motion.

4.4.1.2 Cartesian square Motion

In the second motion the robots are requested to move along a cartesian square which basically lies flat in the 3d-workspace[20]. This motion is less smooth in cartesian space because is includes perpendicular changes of the direction at the corners of the square, but since each motion along a side of the square is brought to a stop before starting along the next side the resulting joint space motion is still sufficiently smooth. But then, depending on where in the workspace the square is placed, the linear motions along the sides of it are more likely to raise a non-linear relation between the cartesian and joint velocities, say, higher joint accelerations may be expected than with the spiral. If a robot only lags behind the desired position in time this will not show up immediately as deviation off the trajectory because the trajectory is a straight line. It will, however, show up as soon as a motion along a new side will be started if the robot has not reached the corner until that time.

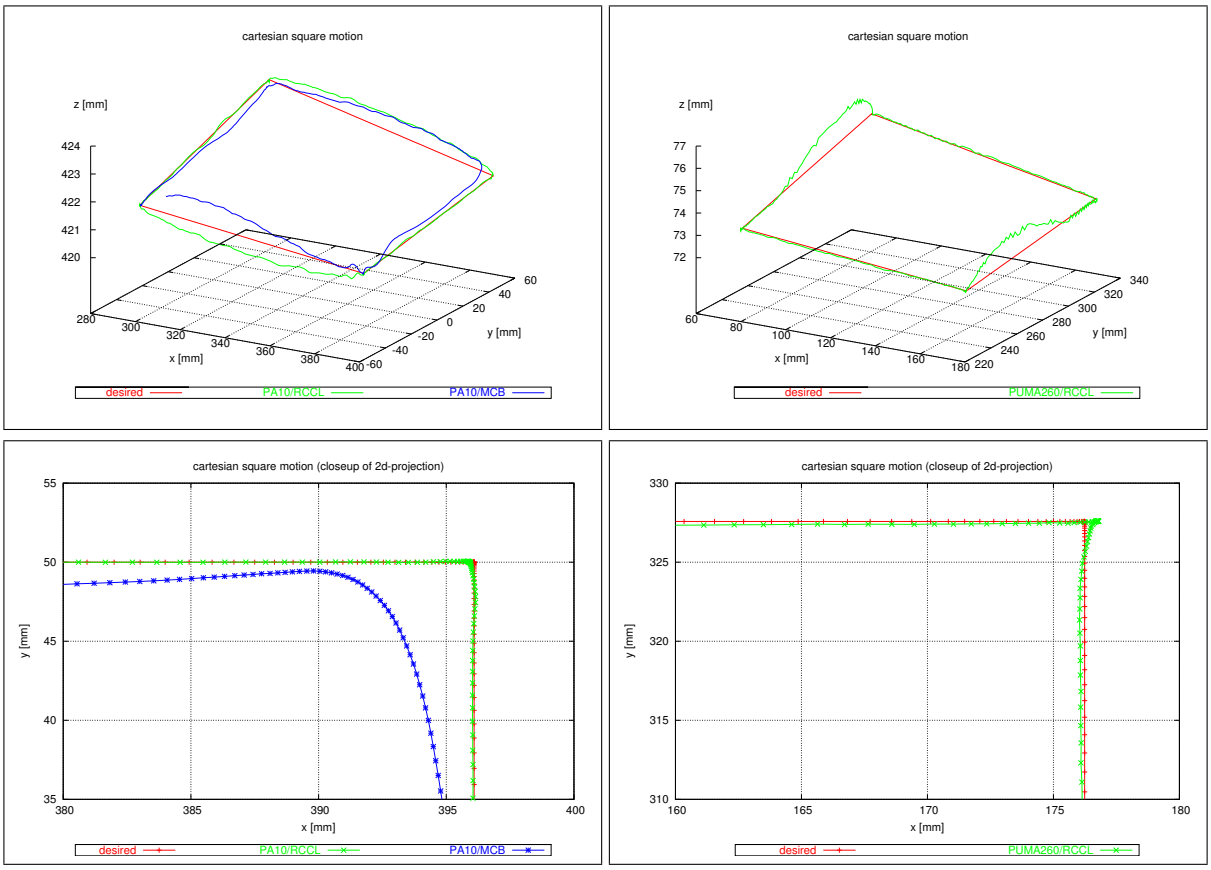


Figure 4.12: Cartesian results from the cartesian square motion. The first row shows the original motion in 3d and the second row shows a 2d projection of the centre of the spiral. The left column shows results from the MCB and RCCL control for a PA-10 and the right column shows results from the RCCL control for a PUMA-260 for comparison.

Figure 4.12 shows the cartesian results from this experiment. In the 3d and 2d-plots of the cartesian position during the motion it can seen that the PA10/MCB solution lags much more

[20]The square is actually not entirely flat in the 3d-workspace, or rather – since the projection *is* a square – it is not really a *square*, but actually only a more general polygon. This does not affect the experiment and is only done to ensure that there is some systematic variance in the z-components of the measurement for better visualisation.

behind than the RCCL versions, but this does not show up so obviously in the 2d-plot while the robot is moving along a side of the square. Only when stopping at a corner and turning around to move along the next side the PA10/MCB solution can be very obviously seen to deviate significantly because it is taking the same shortcuts as with the spiral motion. The 3d-plot, however, reveals that even in those cases where according to the 2d-projection the situation is not so bad a deviation does in fact occur. The PUMA260/RCCL is generally better than both PA10 solutions except in one aspect.

Apparently motions along two opposite sides of the square can be followed quite precisely by both robots (PA10: along y-axis, PUMA260: along x-axis), but motions along the other two sides not (PA10: along x-axis, PUMA260: along y-axis). For the PA10 this effect is less obvious than for the PUMA260 because it is disturbed by a higher positioning noise, but it can still be seen. The reason for this effect is that the kinematical situation is such that the “good” motions translate into joint motions of basically the rotational shoulder joint only, whereas the “bad” motions translate into joint motions of basically the swinging shoulder joint and the swinging elbow joint. Synchronising two joints is more difficult than moving just one joint and the result is an increased error in case where more joints are involved. The reason why this effect occurs at different sides of the square is the different kinematic (placement of coordinate systems) of the PA10 and PUMA260.

Again, the cartesian deviations are caused by a sluggishness at joint level, which is shown in figure 4.13. In the joint plots it can be seen that the PA10/RCCL lags slightly more behind than the PUMA260/RCCL, but not as much as the PA10/MCB. The PA10/MCB yields a MSE of 1.05 degrees with a maximum peak of 3.99 degrees whereas the PA10/RCCL only has 0.18 degrees with a maximum peak of 0.72 degrees. The RCCL solution again performs better than the MCB solution by a factor of more than 5. The performance of the PUMA260 with a MSE of 0.13 degrees with a maximum peak of 0.54 degrees is again the best.

Figure 4.14 shows the euclidian norm of the cartesian displacement between the desired and observed position. The PA10/MCB has a MSE of 15.1 mm with a maximum peak of 34.5 mm whereas the PA10/RCCL only has a MSE of 1.9 mm with a maximum peak of 3.8 mm. This means that again the RCCL solutions performs better by a factor of 8-9. The performance of the PUMA260/RCCL with a MSE of 1.2 mm with a maximum peak of 2.6 mm is again not reached by the others.

4.4.2 Force Control

Another important aspect is the perforce of the robot system when using force control. Force controlled mode is advisable for any motion that is in contact with a fixed object because even by the most advanced vision system the position of these objects can not be determined without error. On the other hand side even the slightest positioning error can cause excessively high forces to occur when the robot/object contact is stiff enough, and in case of the biotechnological devices used in this work this is almost always true. In order to not damage the object and/or cause an operation to fail force control has to be used to compensate the remaining small positional displacements.

Although the PA10 (like many other robots) has a mode where motor torque setpoints can be given this mode is not used for force control. The reason is that it is very complicated to overlay this with an ordinary motion, but such an overlay is required by the majority of force

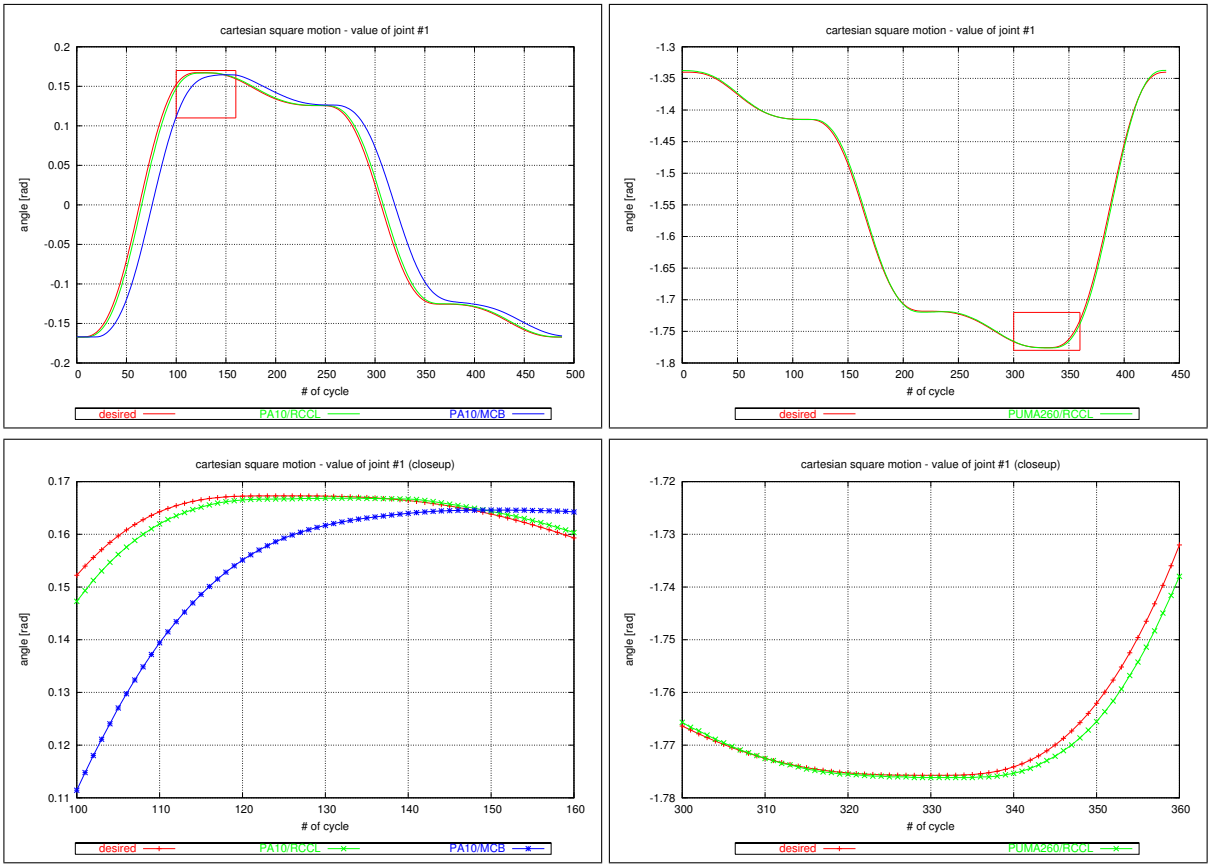


Figure 4.13: Joint results from the cartesian square motion. The first row shows the motion of joint #1 and the second row shows a closeup of the area marked in the first row. The left column shows results from the MCB and RCCL control for a PA-10 and the right column shows results from the RCCL control for a PUMA-260 for comparison.

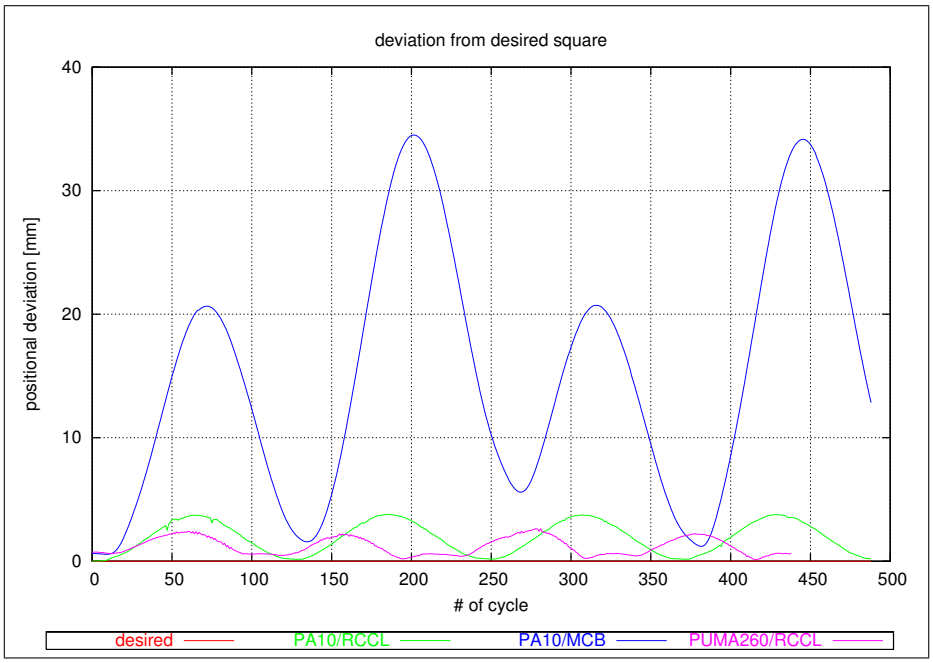


Figure 4.14: Euclidian norm of the cartesian positioning error during a square motion.

control tasks[21]. Except for some special situations which can not be assumed in this work a cartesian dimension can not be mapped to a single joint dimension – the robot's jacobian will generally show that more joints are involved. This means that some joints participate in the force controlled dimension and should be operated in torque mode and the same joints also participate in a cartesian motion in the other dimensions and should be operated in velocity/position mode. This is impossible to do with RCCL and any other readily available software known to the author. Experimental setups that exist at certain research institutes all have to implement an individual robot control software – a task that was not part of the scope of this work.

Force control in this work is therefore implemented as *hybrid force control* like in many other works[22]. With hybrid force control the stiffness of the complete arm mechanism in contact with an object is modelled by a virtual spring hidden in the system. Pressing this spring causes forces to occur, and by moving along the direction of these forces they can be adjusted to desired values by either P(ID) or more complex, non-linear control algorithms like neural networks or fuzzy controllers. This can be done quite easily with RCCL by means of variable transforms, even when superimposed on an otherwise normal motion.

Figure 4.15 shows force plots for the PA10/RCCL when opening/closing the sliding door of the fridge. The f_z force that presses on the door is to be controlled to a constant level of 2 N while a motion is to be done in either positive or negative y-direction. The difference between the two motions can be clearly seen in the f_y forces in the graphs. The f_x forces are perpendicular to the motion direction and the controlled axis and are not meaningful.

The quality of the force control has to be regarded as poor. The forces change with an amplitude of 1 N in a systematic oscillation. This is mainly because the underlying positioning system is too sluggish to follow changes of the position quickly, and so in order to achieve an at least reasonable force control speed the proportional gain of the controller has to be increased almost to a level where the controller becomes unstable. Usage of a more sophisticated and better parameterised controller than a plain P-controller might improve this situation slightly, but will not completely avoid the problem.

A true comparison with a PUMA260/RCCL cannot be done because the PUMA260 is too small to operate the fridge. Instead, figure 4.16 shows force plots for a helix search of a hole on a PUMA260/RCCL. This task is similar in that it moves along (in this case: two) other dimensions than that in which the force is to be controlled, and different in that it "scratches" over the surface rather than hooks behind a door handle and uses a completely different force controller. Yet it serves the purpose of showing that the amplitude of the controlled force of about 0.5 N is by a factor of 2 smaller on the PUMA260 than on the PA10. Even more important is the fact that the characteristic of the force plot is more that of noise rather than that of systematic oscillation. The system is not even near the point of being unstable and in this particular case the proportional gain might even be increased a bit more. Overall, the reaction is much faster and so the motions can be done with higher speed.

[21] For example during opening the sliding door of the fridge the robot should control the force perpendicular to the door because the height of the fridge is not precisely known because it is standing on an uneven base on an uneven floor, but at the same time the robot must make a motion parallel to the door to slide it open.

[22] See [Collani 2001] and [Ferch 2002].

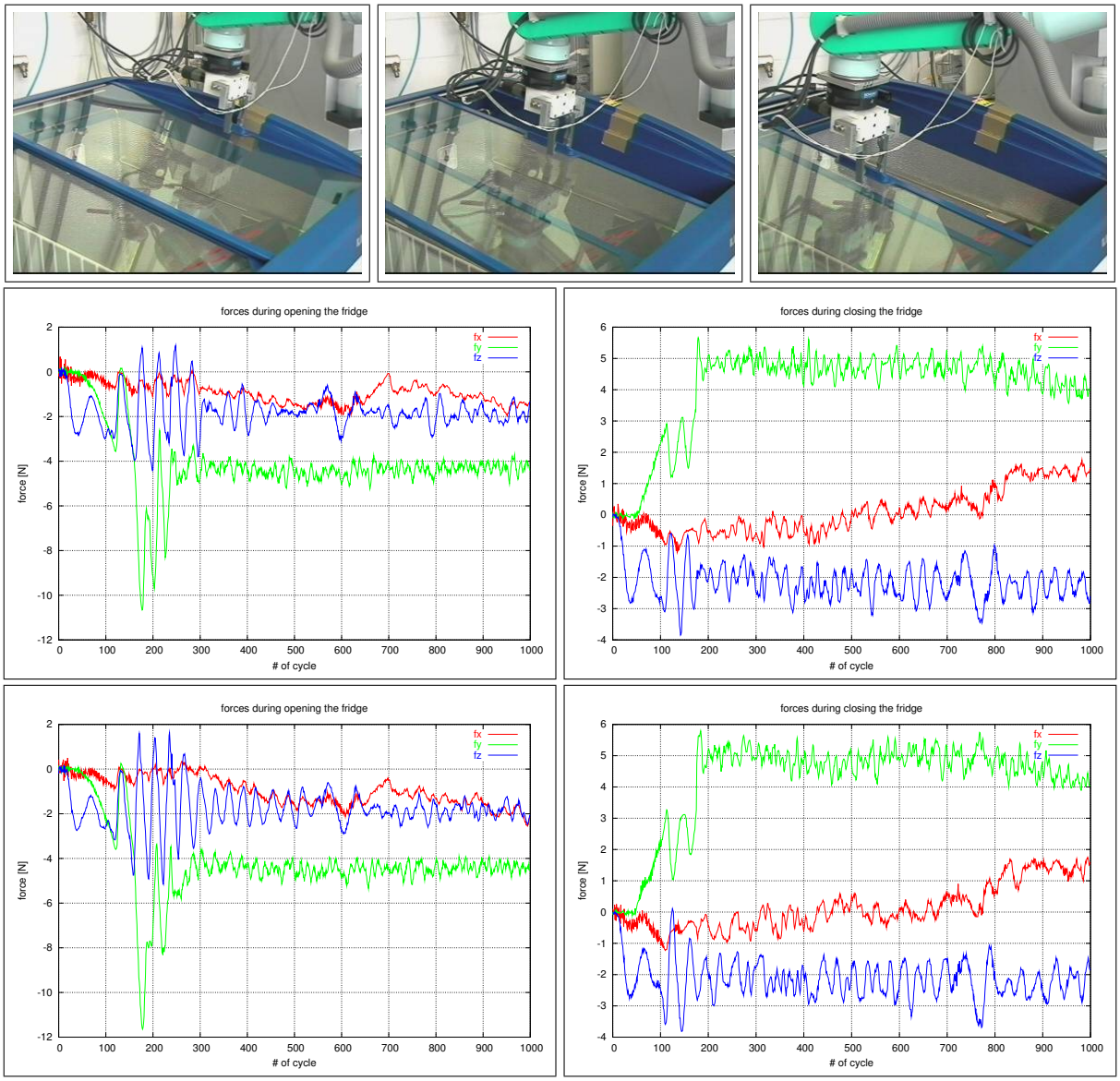


Figure 4.15: The PA10/RCCL doing force controlled motions to open/close of the fridge door. The (green) f_y value shows the direction of the overlayed motion: In the left images it pushes against the door in y-direction to open it and in the right images it pulls the door along the negative y-axis to close it. The (red) f_x value is perpendicular to the motion direction and presents no relevant information. The (blue) f_z value is to be adjusted to -2 N so that the finger of the robot tool presses with 2 N on the door while moving it.

4.4.3 Summary

The results prove that using RCCL on a PA10 is a feasible approach. The resulting robot control system is not as reactive as a PUMA260 or any other position controlled robot and special care has to be taken in some cases, but it performs better in all tested aspects than the PA-Lib by MHI running on the MCB.

The performance results of the positioning are summarised in table 4.3 for the joint values and in table 4.4 for the cartesian values. The average displacement of about 1 degree and peak displacements of about 4 degrees at the joint level of the of the PA10/MCB indicate a very

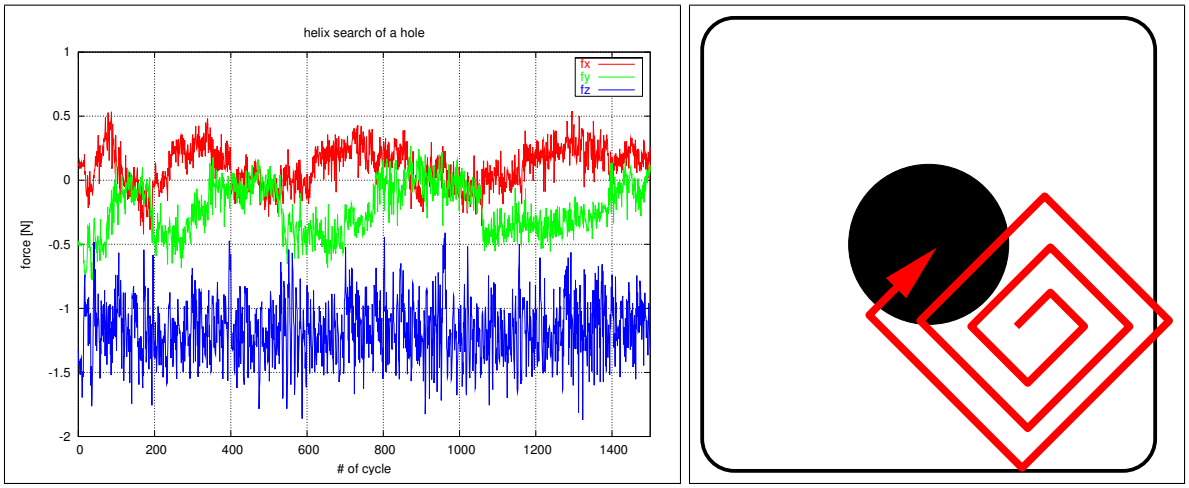


Figure 4.16: The PUMA260/RCCL doing a force controlled helix search of a hole in a planar object.

sluggish response of the system. Tests of responses to step functions that are not given here show that it takes about 0.3 s to reach the desired value where a PUMA260 is almost 10 times faster. The reason for this is not that the PA10 has several time the mass than the PUMA260, but simply that the MCB implements a *very* sluggish TGEN. Using RCCL and an additional joint controller improves the joint behaviour by a factor of about 5.

	PA10/RCCL			PA10/MCB			PUMA260/RCCL		
	min	avg	max	min	avg	max	min	avg	max
spiral	0.0000	0.2219	0.7720	0.0013	1.1412	3.7925	0.0000	0.1300	0.3447
square	0.0000	0.1816	0.7216	0.0007	1.0486	3.9853	0.0001	0.1368	0.5414

Table 4.3: Displacement between the commanded and observed values of joint #1 during test motions for all three tested robot setups. All values are in degrees.

The sluggishness at the joint level naturally corresponds to a sluggishness at the cartesian level. With average displacements between 15 and 18 mm and peak values between 34 and 46 mm the PA10/MCB cannot be said to be usable for fine-positioning. Displacements in that magnitude not only mean that the paths will be notably distorted, but also that collisions with objects become possible – the very avoidance of which is the reason to do path-planning in the first place. Using RCCL and an additional joint controller improves the cartesian behaviour by a factor of almost 8.

	PA10/RCCL			PA10/MCB			PUMA260/RCCL		
	min	avg	max	min	avg	max	min	avg	max
spiral	0.018	2.376	4.949	0.595	18.333	46.041	0.037	1.336	2.519
square	0.015	1.904	3.782	0.562	15.172	34.503	0.138	1.203	2.622

Table 4.4: Displacement between the commanded and observed cartesian position during test motions for all three tested robot setups. All values are in millimetres.

As a consequence of the suboptimal performance of the PA10/RCCL all the force controlled motions have to be executed with rather slow speed in order to not become unstable. With the current hardware there is no way to overcome this drawback. The newer PA10-6C/PA10-7C have a cycle time of 2 ms which might theoretically allow to run the PID controller 5 times faster than RCCL's TGEN without changing that one's cycle time of 10 ms, though due to the internal thread structure of RCCL this will be quite complicated. Whether this really improves the positioning performance has to be left an open question because it has neither been part of this work, nor had the necessary resources for the tests been available.

The PA10/RCCL system established in this work *is* capable of performing the required tasks according to section 2.3, although the performance of the newly implemented PA10/RCCL solution lies below that of a 20 year old PUMA260. With the initially intended PA10/MCB solution the tasks would not have been possible. Using RCCL therefore has to be seen as the proper choice.

Chapter 5

Robust Colour Vision

"If whatever I'm supposed to be looking for is here, I don't want it."

Zaphod Beeblebrox

In this chapter the colour vision system used on the robot arm is presented. This vision system is a crucial part of robot system because it is the arm's most important sensor used to detect an object's positions before manipulating it. The arm needs such a sensor because it can not assume the teached positions to be accurate.

As mentioned in chapter 3 the positioning of the mobile platform is not absolutely accurate. This inaccuracy is inherently enhanced by the two explicit goals of the robot system that the setup should be easily modifiable and that all devices should remain usable by human personnel for other tasks when the robot is idle. As a result the positions of the devices can not be assumed as fixed. Human personnel can and will change the positions more or less slightly by touching the devices, even if only accidentally. Therefore – even if the mobile platform's positioning was perfect – it can only be considered to bring the arm "close" to an object, but not close enough so that it can manipulate the object without further sensing.

A newly designed colour vision system is used to detect and compensate these positioning errors. Since the mobile robot system operates without human supervision this vision system has to be absolutely reliable. In the car manufacturing industry a *driver assistance system* (DAS) for detection of pedestrians is considered reliable if it reaches detection rates of 99%[1], simply because the system only assists the driver, who – still remaining in full responsibility – will detect the remaining 1%. For a mobile robot system in a laboratory a success rate of 99% means that it is not usable, because with no supervision there must not be any error at all.

Usually a system that is supposed to work without any error requires all environmental parameters to be under perfect control, however, a real world environment like a biotechnological laboratory for example means to not have a standardised illumination – a condition that is often required by computer vision. The vision system presented in this work therefore has to tolerate a wide range of illumination intensity. It addresses this problem by using that component of vision that is least sensitive to the illumination intensity – colour information.

[1] c't magazin für computer technik 17/2004, p.76, Heise Verlag, Deutschland.

The colour segmentation proposed in this work allows for a very reliable detection of image features, and the model based object recognition further enhances the reliability. The system can still fail under extreme circumstances, but contrary to many other vision systems – especially learning based systems – the limits can be explicitly given and the failure can not go unnoticed. Therefore, the system has achieved a so far unprecedenced success rate as defined by not recognising errors.

The vision system consists of a microhead colour camera mounted to the arm tool and used to grab colour images. In these images the objects are detected by extracting coloured regions that form individual patterns for each object. These patterns are classified and their position in the image is computed. Once the position is known the arm can apply correctional motions to centre over the object and thus learn about the object's position in world coordinates. The only prerequisite needed for successful operation is that the positioning of the mobile platform and the arm must be good enough to ensure that the object is within the initial viewing area of the camera.

The structurisation of this chapter is as follows: Section 5.1 discusses the foundation of the vision system, the motivation why colour vision is used, the hardware used by the system and its software structurisation. This system basically falls into three different parts: Section 5.2 covers the image segmentation part in which the dimensionality of the information is reduced from that of a complete colour image to that of a list of coloured regions. Section 5.3 covers the object recognition and displacement computation part in which the region lists are searched for patterns of known objects as well as the position of a thereby detected object in the image is computed. Section 5.4 covers the displacement compensation part in which the detected object is centred in the image by means of moving the camera (arm). Finally, section 5.5 gives the results of some experiments to show the system's performance, summarises the most important aspects and gives an outlook into possible future work.

This chapter uses a lot of terms and constructs from computer vision and classification, the basics of which are assumed to be known. Nevertheless, a short introduction into some terms and ideas is given in section C.2 in the appendices.

5.1 The Implemented Colour Vision System

In this section an overview and introduction into the established colour vision system will be given: Subsection 5.1.1 gives the motivation for using colour vision, subsection 5.1.2 briefly describes the vision hardware and subsection 5.1.3 describes the software structurisation.

5.1.1 Why Colour?

The centre-point of the design of the vision system is the decision to use a colour based approach. This is an important point because most contemporary vision system are still based on greyscale imaging only. With colour images the amount of information is three times as high as with greyscale images, and accordingly the processing requires more resources. Likewise, most of the known basic image manipulation techniques have originally been defined for greyscale images only. Applying three instances of such a filter to the three channels of a RGB image is no technical problem, but does not automatically yield new results. So the question is: Why colour?

The reason is the one dominating problem of greyscale vision – the illumination. By looking only at the brightness of pixels it can be impossible to tell whether a bright spot caused by a feature of the object or because of some spot of light falling on it. This is particularly troublesome when a later processing step applies a binary threshold to the image, as is often the case. Figure 5.1 shows an example of such a situation. An image like this is extremely troublesome for a greyscale computer vision system because it can not be guaranteed that a binary threshold to segment the water cooker from the background can be found.

Figure 5.1: A bright object in front of a dark background with sunlight falling through a blind. When using binary thresholds on greyscale images the object is almost impossible to segment.

Because of this problem most greyscale computer vision systems require a normalised illumination. In a computer vision laboratory this can usually be accomplished without major problems because the environment can be defined to fit to the vision system. In a real-world laboratory normalised illumination can not generally be guaranteed because only limited or no modifications of the environment may be tolerable. The obvious idea to attach lamps to the robot is not so trivial as it would then have to be guaranteed that these lamps dominate all other light sources but do not cause systematic brightness gradients in the image. Over all, the most promising approach is to make the vision system tolerant against a wide range of changes of the illumination.

The idea to use colour information to make the vision system tolerant against changes of the illumination was first inspired by the coloured caps of the NUNC tubes and the coloured button of the centrifuge as in figure 5.2. This approach is feasible and justifiable because all objects are either coloured in the first place, or can easily be tagged with coloured labels, the latter of which being only a neglectable modification of the system's environment. It is well-known that colour itself changes much slower than the brightness with varying illumination. In the RGB colour space this is not so obvious, but the YUV and HSV colour spaces are downright

designed for this because they explictly separate *colour* from *brightness* (see section C.2.2 in the appendices). By employing this decoupling illumination influence can be ruled out to a large extend.

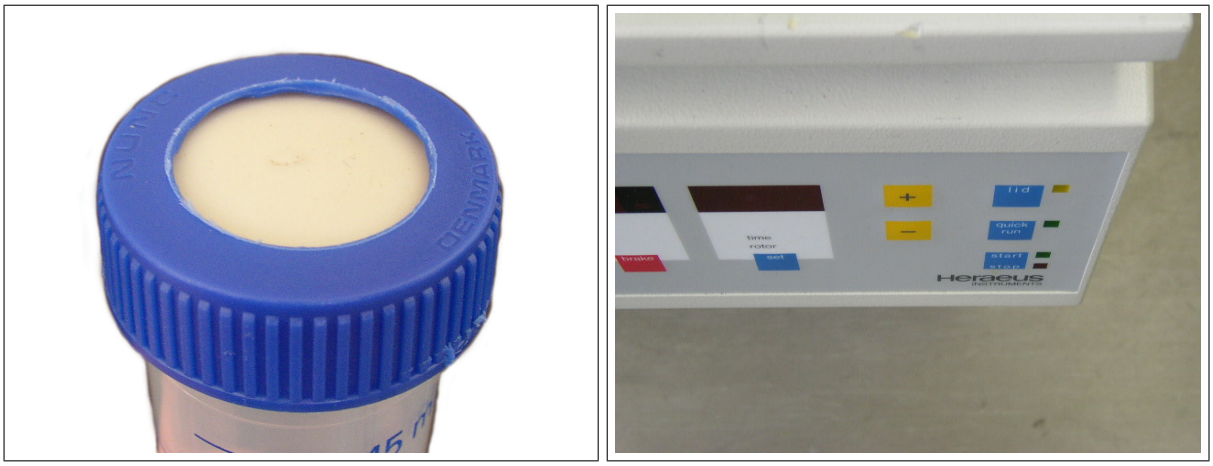


Figure 5.2: The blue cap of the NUNC tubes and the coloured buttons of the centrifuge have inspired the usage of a colour vision system.

For example, the gray value of the yellow buttons of the centrifuge in figure 5.2 is already with normal illumination quite similar to the gray value of the front panel around it. If the brightness of this image is gradually decreased and – as common with dark images – a little noise added, the buttons quickly become difficult to see in greyscale images. Figure 5.3 shows clearly that with decreasing brightness it becomes more and more difficult to make out the yellow buttons in the V channel, but they still can be seen in the S and in particular in the H channel of a HSV image. This means that colour is an important information that should not be discarded for the mere sake of simplicity of the algorithms.

5.1.2 Vision Hardware

The hardware used in the vision system consists of two parts: A colour camera and a colour framegrabber board, both of which were state-of-the-art at the time of the project start.

The colour camera is mounted at the wrist of the arm tool. Since it must be very small in order not to obstruct the arm movements a microhead (lipstick) camera is used. The camera delivers raw signals to a receiver box that controls all camera parameters like gain and shutter values. The box then delivers an analog video signal as either analog colour composite FBAS, SVHS or RGB.

Due to the camera size all of its components must be very small, including the CCD sensor. A small sensor means that amplification is needed because it does not catch very much light, and amplification adds noise to the signals. Because of this noise level and because of the fact that the signals are transported quite a long way on an analog cable the resulting image quality can not compete with current digital still cameras. The comparison in figure 5.4 show rather drastically that especially low saturated colours are almost completely suppressed on the M1250.

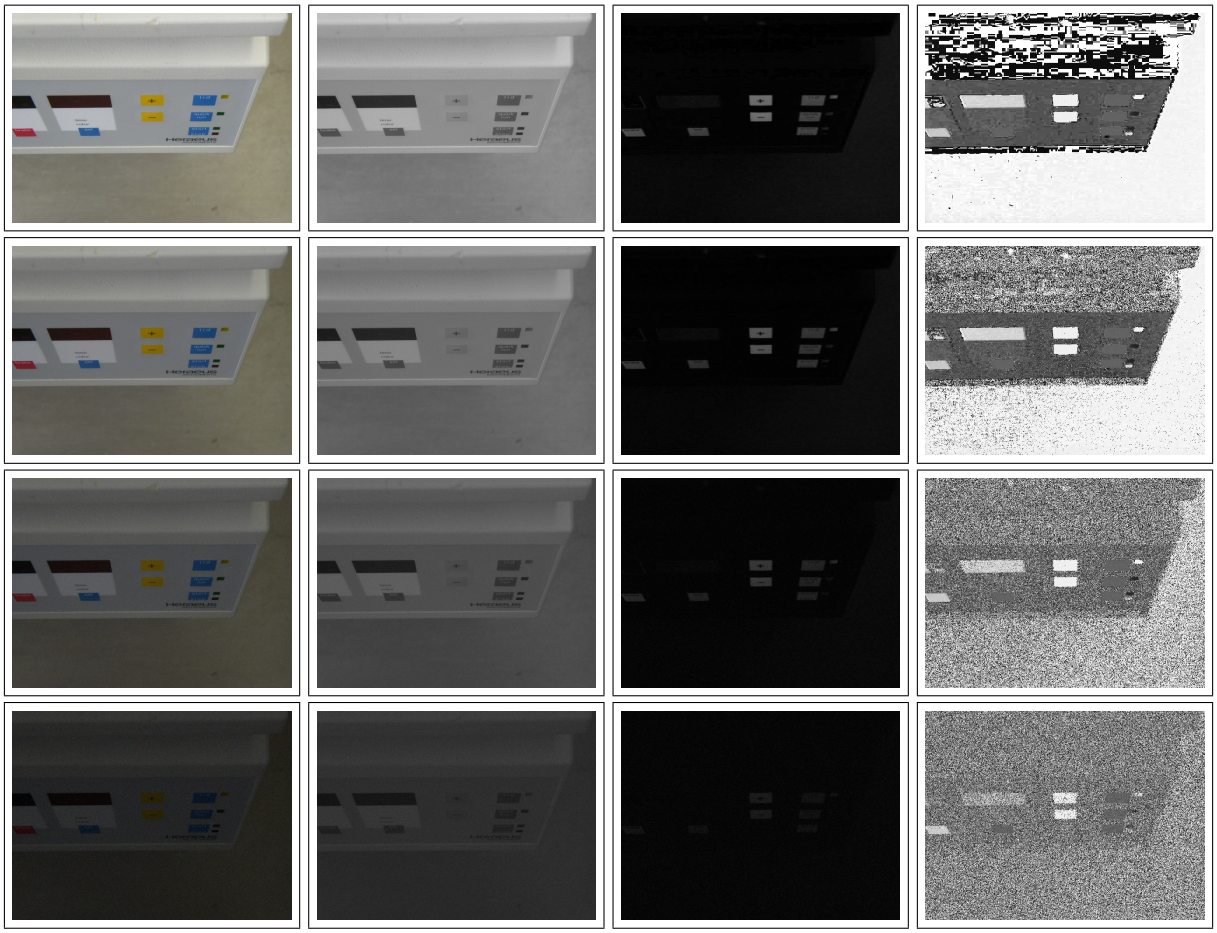


Figure 5.3: The coloured buttons of the centrifuge under different illumination conditions. The columns from left to right show the original image and the V, S and (non-colourised) H channels of the HSV decomposition. The rows from top to bottom show situations with 100, 75, 50 and 25% illumination and increasing noise level. Where with decreasing brightness it becomes more difficult to make out the yellow buttons in the V channel they can be more easily seen in the S and in particular in the H channel.

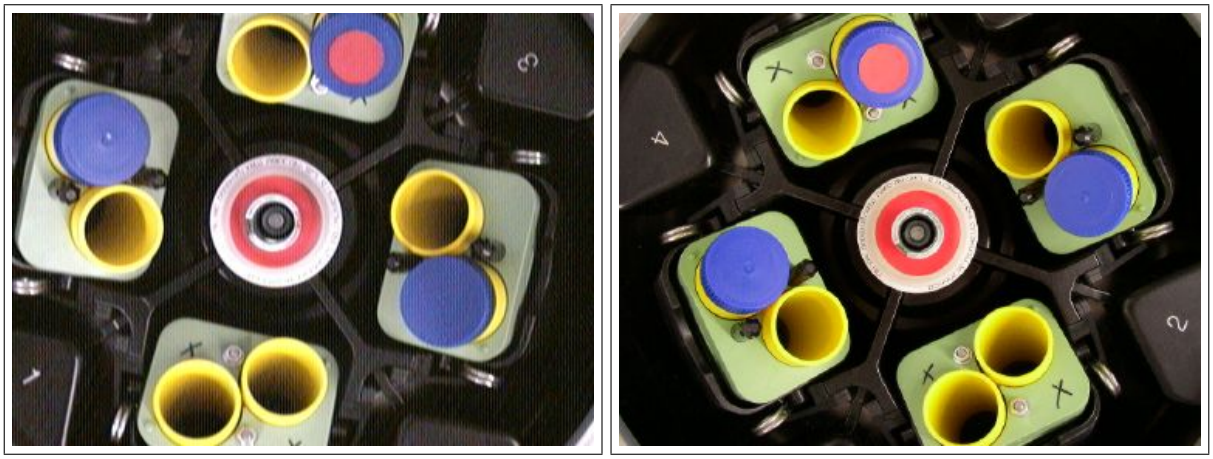


Figure 5.4: The colour saturation of the JAI M1250 microhead camera (left) compared to an Olympus C-300 Zoom digital still camera (right).

This situation is even worsened by the need to digitise the analog signal data with a colour framegrabber board, which is used to deliver YUV colour images to the software. Even though the SVHS cable used does provide a better signal quality than a FBAS cable it still adds to the signal noise, especially because of interference of other electric components on the mobile platform. The image quality resulting from the suboptimal camera, the analog transmission to and the digitisation on the framegrabber board is rather poor, as can be seen in figure 5.5. Vision software operating with these images has to be tolerant against these disturbances.

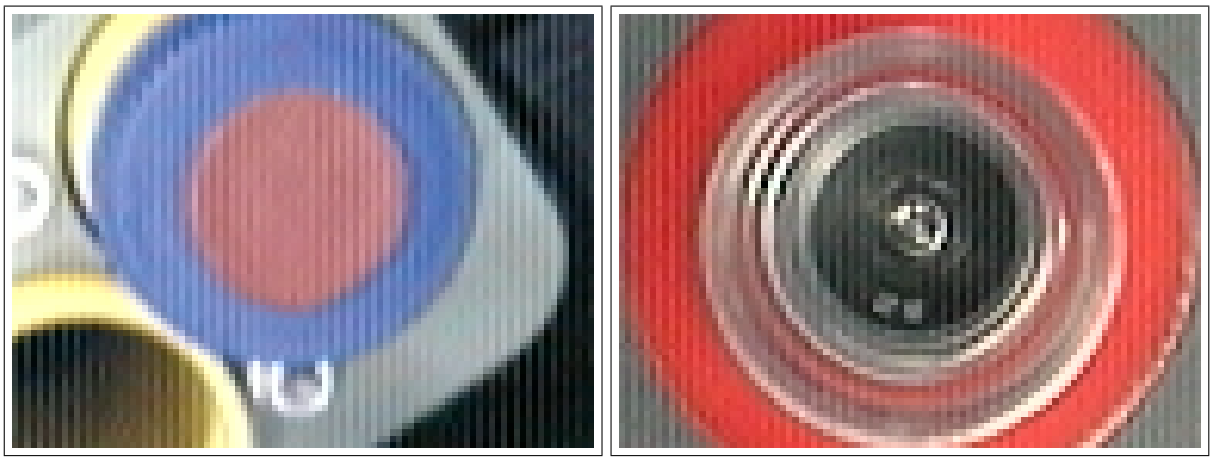


Figure 5.5: Closeup of sample images demonstrating the poor image quality of the vision system hardware.

For a more verbose description of the vision hardware see section C.1 in the appendices.

5.1.3 System Structurisation

The software to work on the hardware is split in two major parts, The CIMAGE library operating at image level and the CVISION software operating at object level.

The CIMAGE library operating at image level deals with low-level vision aspects like image grabbing, manipulating, displaying and storing. The image manipulation capabilities include colour space conversions, different filters, image resizing and some drawing capabilities. To allow manipulations not implemented in CIMAGE the raw data can be easily accessed.

The CVISION software operating at object level uses this CIMAGE library as a foundation for its work. It implements the high-level vision process as shown in figure 5.6 and described in the following sections. This vision process is in turns split in three parts:

- The *image segmentation* stage uses the CIMAGE library to extract a list of coloured regions from the image. It is presented in section 5.2.
- The *object recognition & displacement computation* stage uses this list of coloured regions to generate models of objects, recognise objects according to these models and compute their position in the camera coordinate system. It is presented in section 5.3.
- The *displacement compensation* stage uses small, iterative motions of the robot arm to correct the object positions. It is presented in section 5.4.

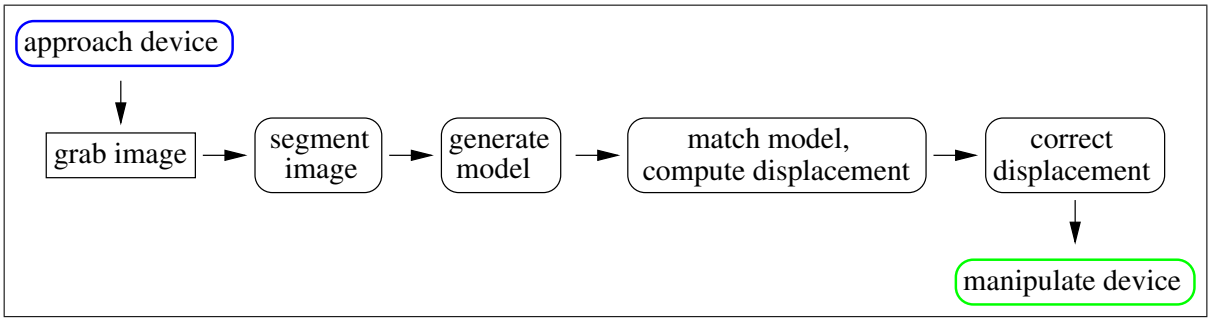


Figure 5.6: Execution flow of vision process.

5.2 Image Segmentation

The first step of the vision process is to extract a minimal set of characteristic features fully describing the situation in order to reduce the problem's dimensionality. Coloured regions are chosen as features because the image background of the possible situations is mostly colourless and the colour component changes only very slowly with changing illumination. A colour similarity measure is defined to allow the comparison of colours. Using this measure the images are segmented by a flood fill algorithm into regions of similar colour, resulting in a list of only the *centre of gravity* (COG) and colour of the regions. This feature list is used to recognise objects and compute their position in the camera coordinate system as described in section 5.3.

Starting with a raw YUV image the steps of the segmentation process are as in figure 5.7. After splitting the image into separate Y, U and V channels the U and V channels are smoothed with a new smart smoothing filter (subsection 5.2.1). Optionally, edges in the Y image can be used to enhance borders between regions in the U and V image. A colour is then chosen from the UV plane according to one of two possibilities (subsection 5.2.2.1 and 5.2.2.2). If no proper colour has been found the segmentation terminates. If a proper colour has been found a similarity measure of image colours to the chosen colour is computed (subsection 5.2.2). In this similarity measure the most similar pixel is chosen (the fill seed). Around this pixel a region is filled using a flood fill algorithm (subsection 5.2.3). The region is removed from the image and the process repeated by searching for a new colour in the UV plane.

In the following these segmentation steps will be demonstrated at the image of the centrifuge in figure 5.8.

5.2.1 Smart Smoothing

The first step after image acquisition is to smooth the image to remove noise. This is necessary because the generally poor signal quality of the camera, the analog transmission of the video signals and the grabbing have already at this first stage deteriorated the image quality significantly. This manifests itself in the fact that in order to not destroy even more image contents classic smoothing algorithms can not be used.

Classical, convolution based linear filtering (see section C.2.3 in the appendices) processes the image by running a mask (the filter kernel) over each pixel of the image. In case of the *mean* filter it then computes the average pixel value p_μ of the pixels inside this mask and assigns this as the new pixel value p_{new}. As such, the mean filter smoothes outliers from the image, but

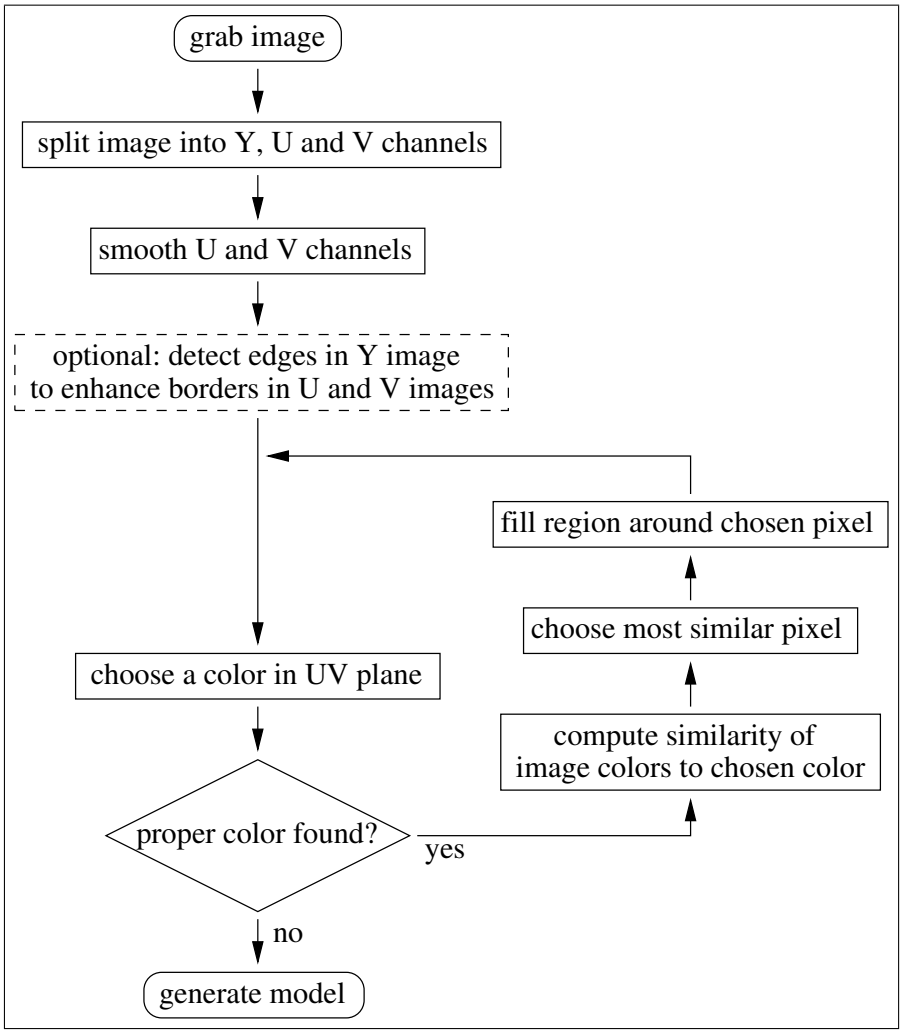


Figure 5.7: Execution flow of image segmentation.

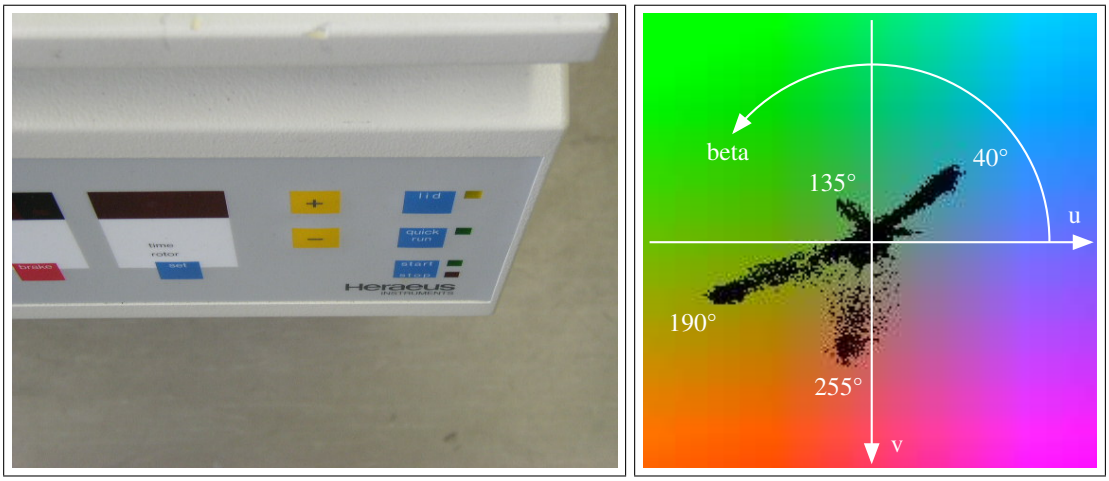


Figure 5.8: A sample image of the centrifuge. The blue (β = 40°), yellow (β = 190°) and red (β = 255°) buttons can be clearly seen as fingers in the UV colour plane usage. The green (β = 135°) finger that can also be seen belongs to LEDs on the front panel. It is too unstable to contribute to the recognition.

does not remove them completely. Also, it does not differ between noise and meaningful image contents, therefore it blurs the entire image.

Examples for the mean filter can be seen in figure 5.9. The source image has deliberately been chosen to be noisy. As can be seen, by averaging over the pixels in the filter kernel the filter effectively suppresses noise, but also blurs sharp edges. In order to smooth images but not overly blur edges other filters are therefore needed.

To overcome the disadvantages of the ordinary *mean* filter a *smart mean* ("smean") filter has been developed for this work. The idea with non-linear "smart" techniques (see section C.2.4 in the appendices) is to take additional information into account that can not be expressed linearly. In case of the smean filter this means that smoothing should only be applied to those areas of the image where it can be expected to be helpful. For example, areas uniformly filled with a colour but slightly distorted by noise should be smoothed, but edges between areas of different colours should not.

The smean filter is similar to the mean filter in that it uses the same kernel to compute the average value p_μ of an environment of pixels, but differs in the step that comes afterwards. Where the mean filter simply assigns this average value p_μ to the new pixel p_{new} the smean filter also computes the deviation σ of the environment of pixels. This σ expresses how "equal" the pixels in the filter kernel are. It is low when the pixels are similar and high when they are not, as in case of an edge. Together with a scaling factor c it is then used to weight the average and the original pixel value to compute the new pixel value as

$$p_{\text{new}} \quad = \quad e^{\frac{-\sigma}{c}} \cdot p_\mu + (1 - e^{\frac{-\sigma}{c}}) \cdot p_{\text{orig}}$$

Figure 5.10 shows examples of the smean filter. Compared to the mean filter in figure 5.9 it can be seen that the smean filter only blurs those areas of the image where pixels have already been somewhat similar, but maintains rather sharp edges. This feature makes the smean filter very interesting for smoothing a YUV colour image where the UV colour plane has a rather low signal/noise ratio but edges should be maintained by all means.

5.2.2 Colour Similarity

The primary intention with using colour information is to use its additional information to gain better and more robust vision performance, however, this intention leads to a triplication of the amount of data that has to be handled. Since a lot of algorithms are easier to use and perform better when they are applied to only one-dimensional data it is therefore desirable to reduce the dimensionality as early as possible. The image segmentation algorithm implemented in this work allows exactly that because it uses an abstract one-dimensional similarity of colours.

In the YUV images used in this work a pixel's brightness is represented by a 1-dimensional value y along the Y axis, whereas its colour is represented separately by a 2-dimensional vector $\vec{c} = (c_u, c_v)$ in the UV plane. The centre $(0, 0)$ of that plane represents all grey pixels from black to white. The naive approach would be to use the euclidian distance

$$d(\vec{c}_1, \vec{c}_2) \quad = \quad \|\vec{c}_1 - \vec{c}_2\|$$

as a similarity measure,. However, this would neglect an important feature of colour, the distinction between colour *hue* $\alpha(\vec{c})$ and colour *saturation* $s(\vec{c})$ as per

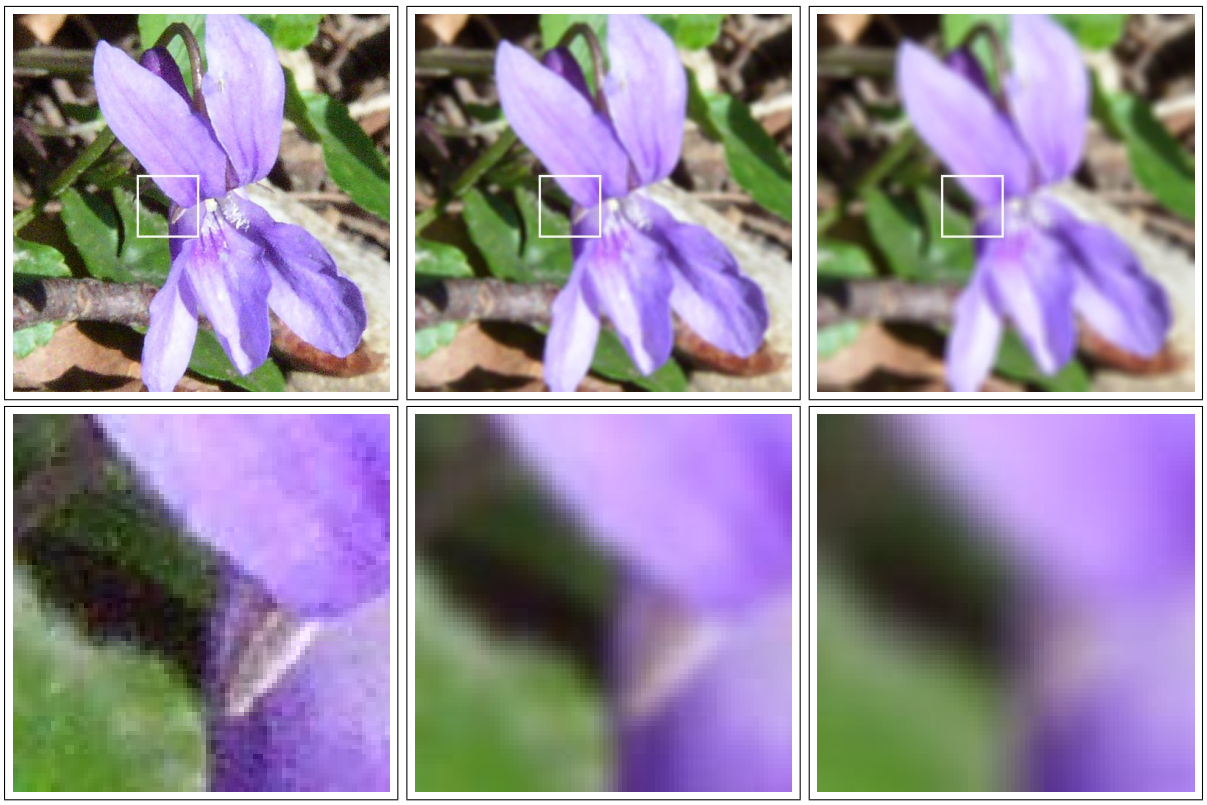


Figure 5.9: The mean filter smoothing an image. Upper row from left to right: Original image, 7×7 mean and 15×15 mean. Lower row: close-ups of the areas marked in the upper images.

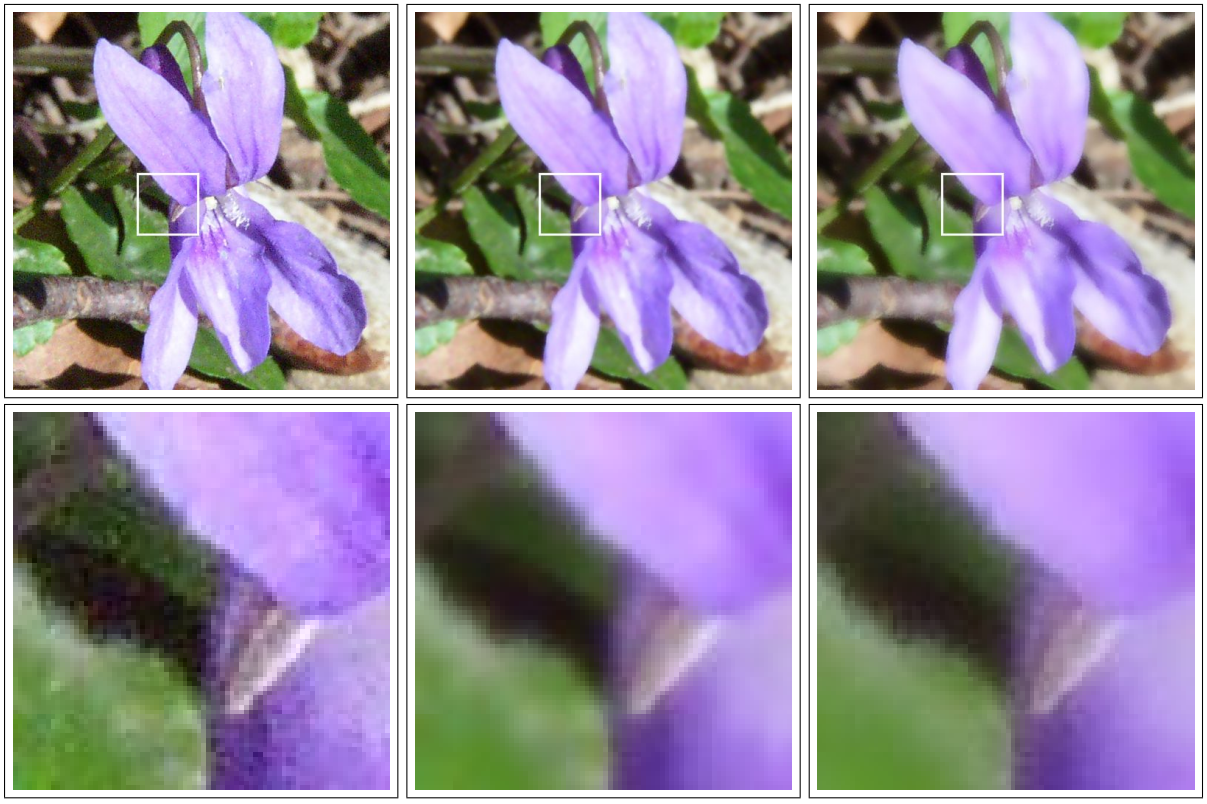


Figure 5.10: Smart smoothing of an image. Upper row from left to right: Original image, 7×7 smean and 15×15 smean. Lower row: close-ups of the areas marked in the upper images.

$$\begin{aligned} \alpha(\vec{c}) &= \arctan \frac{c_v}{c_u} \\ s(\vec{c}) &= \sqrt{c_u^2 + c_v^2} \end{aligned}$$

The second naive approach to use euclidian distances on α and s separately is also not advisable because the objects in this work have colours consisting of a wider range of hue/saturation values. A more sophisticated measure is therefore needed.

The basic idea behind this measure is to weight the differences in saturation s and hue α differently. Given a desired colour hue β, that is, a vector $\hat{c}_\beta$ in the UV space[2] the saturation similarity d_s is computed as the projection of a colour $\vec{c}$ onto the desired colour $\hat{c}_\beta$.

$$d_s(\vec{c}, \hat{c}_\beta) = g\left(\vec{c} \cdot \hat{c}_\beta\right) \tag{5.1}$$

The function g is used to address two problems:

1. The scalar product $s = \vec{c} \cdot \hat{c}_\beta$ can become negative if complementary colours are used[3]. Such a negative similarity is undesired and has to be avoided. Of course, clipping at zero as per "$s = (s > 0)\ ?\ s : 0$" in C/C++ notation would be the trivial way, but would still ignore the second problem.

2. Since the vision system is explicitly designed for coloured objects a similarity only makes sense for saturated colours. In case of pale colours with a saturation of less than some s_0 any similarity should therefore explicitly be suppressed.

Both these problems can be eliminated by applying a non-linear scaling of s. A multitude of possible scaling functions g exists. For example, $1 - \exp(-s + s_0)$ yields a scale factor that brings down the output to zero for inputs near s_0, but becomes negative for even smaller inputs. Multiplied with an already negative input the total output increases exponentially. The function is theoretically applicable, but needs additional clipping and is therefore not recommendable. Instead, the function

$$g(s) = \frac{\tanh(s - s_0) + 1}{2} \cdot s$$

or, if the transition should be smoother, with an additional factor $f < 1$ as

$$g(s) = \frac{\tanh(f \cdot (s - s_0)) + 1}{2} \cdot s$$

more smoothly mapping $[-\infty..\infty] \to [0..\infty]$ is used. The term $\frac{\tanh(s)+1}{2}$ yields a scale factor that converges to 0 for $s \to -\infty$ and 1 for $s \to \infty$ with an inflexion point at $s = 0$. Using the s_0 parameter the inflexion point can be arbitrarily moved and using the f parameter the

[2]The notation $\hat{c}_\beta$ means that β is a colour with full saturation. The resulting vector is therefore a unit vector for the colour β.

[3]Note that the variable s is used here for the similarity of two colours as well than the saturation of a single colour. The similarity can always be expressed as the saturation of the vector $(\vec{c} \cdot \hat{c}_\beta) \cdot \hat{c}_\beta$ resulting from the scalar product, hence the double usage.

steepness of the transition can be modified. This way large saturation similarities[4] are nearly linearly passed and small (or negative) ones are suppressed. Plots of the functions can be seen in figure 5.11.

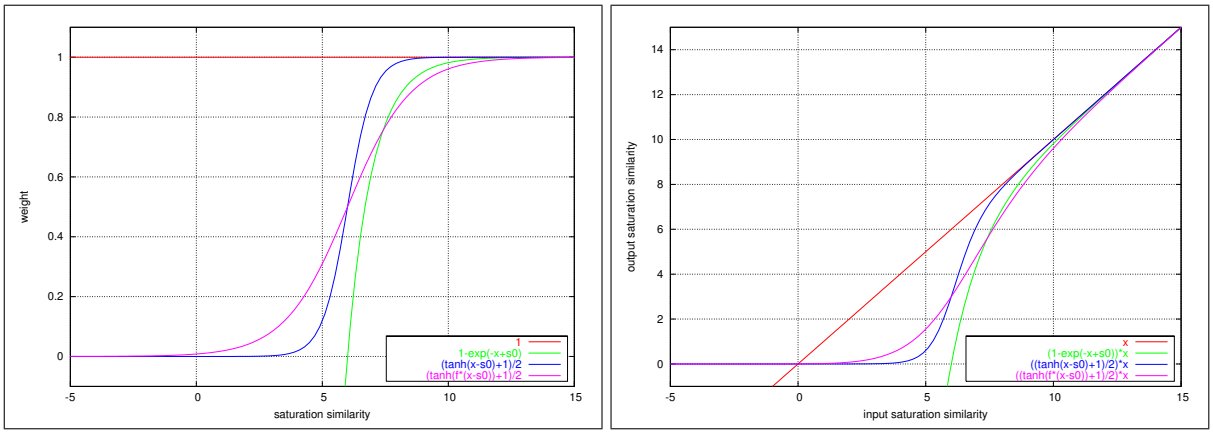


Figure 5.11: Non-linear scaling of the saturation similarity. Left image: weight factor implemented by the scaling function, right image: weight factor applied to the saturation. The parameters used are $s_0 = 6$ and $f = 0.4$. The operations works in an abstract integer domain, so the scales are not ± 1.

For example, the centrifuge in figure 5.8 consists of blue (β = 40°), yellow (β = 190°) and red (β = 255°) buttons. The saturation similarity from equation 5.1 applied to the colours of this image yields weights as in figure 5.12 (contrast-maximised to match the printable range of grey values).

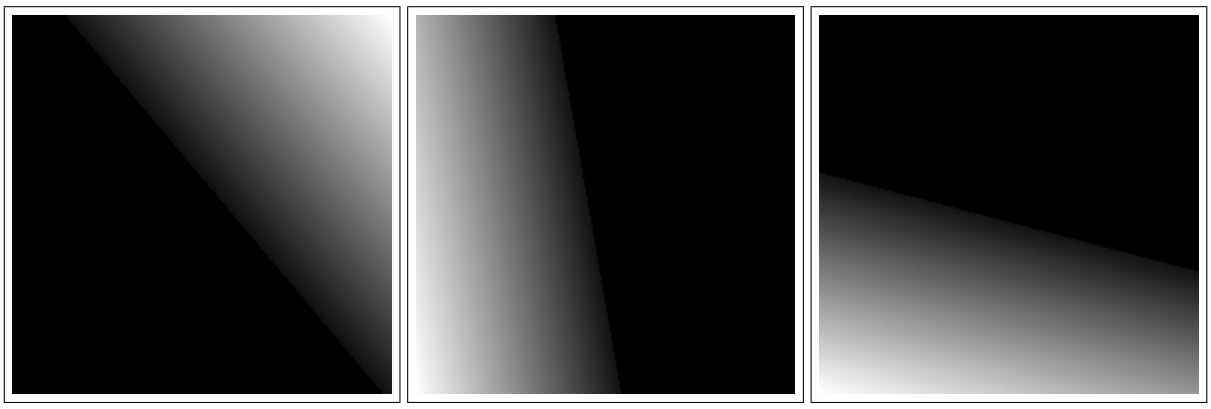


Figure 5.12: The saturation similarity from equation 5.1 applied to the colours of the centrifuge.

The hue similarity d_α is the second part of the total similarity. It is computed as the absolute value of the projection of a colour $\vec{c}$ onto the vector $\hat{d}_\beta \perp \hat{c}_\beta$, expressing the distance[5] of the

[4] Actually, this term might appear a bit irritating because the similarity only compares the saturations s of two colours if the colour hues are close together, that is, the vectors $\vec{c}$ are nearly collinear. The more non-collinear the vectors are the more it is the hue α that affects the measure. Nevertheless it is called saturation similarity here because the primary purpose of this measure is to separate colours of similar hue according to their saturation.

[5] This means that colours of low saturation are allowed to change more than colours of high saturation, but this is only a side-effect. It is, however, a desired side effect because low saturated colours are usually affected by noise too, and in this case it might be of advantage to be a bit more tolerant compared to high saturated colours.

colour $\vec{c}$ from the line generated by the $\hat{c}_\beta$.

$$d_\alpha \quad = \quad h(|\vec{c} \cdot \hat{d}_\beta|) \tag{5.2}$$

Again, a function h is required to address two problems:

1. The scalar product $s = \vec{c} \cdot \hat{d}_\beta$ yields a similarity that changes linearly over the input range. With the saturation similarity this is acceptable, but with the hue similarity it is not. One result would be that strongly different colours are still weighted too high, which must be avoided.

2. The scalar product yields an inverted range of values, that is low values correspond to good similarity and high values to bad.

Again, both these problems can be eliminated by applying a non-linear scaling of s and again a multitude of possible scaling functions h exists. For example, $exp(-4s/s_{\max})$ yields a non-linear conversion of s that solves both problems, but allows only a very small range of input similarities to be weighted high. It is more selective towards exact matches than desirable, and therefore not recommendable. Instead, the function

$$h(s) \quad = \quad \exp\left(-\left(\frac{s}{2s_{\max}}\right)^2\right)$$

more smoothly mapping $[0..s_{\max}] \rightarrow [1..0]$ is used. The term $\exp(-s^2)$ yields a result that smoothly becomes 1 for $s \rightarrow 0$ and 0 for $s \rightarrow s_{\max}$. Using the maximum similarity $s\max$ the output range can be adjusted to the input range because the measure is designed to be relative to the extend of the input range that is actually used[6]. If only a small band of the theoretically possible input range is used because all existing colours are of very similar hue the measure should be more strict in segmenting the hues. Plots of the functions can be seen in figure 5.13.

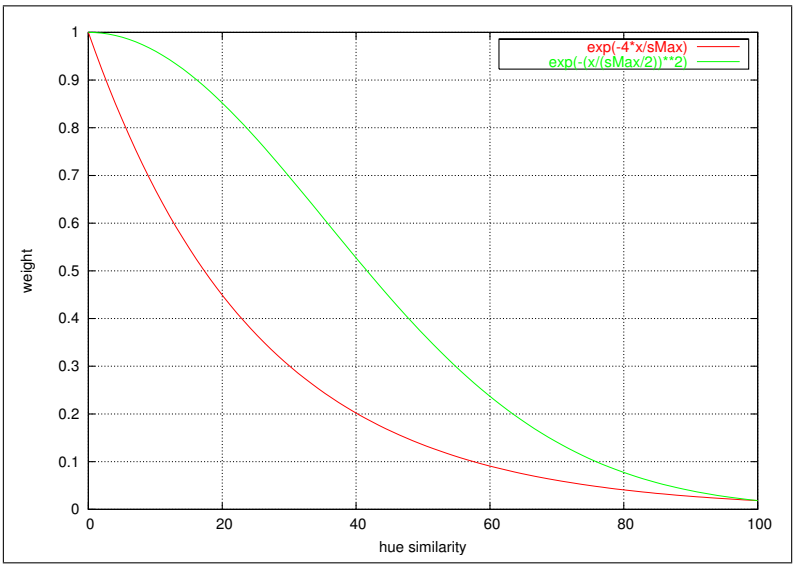


Figure 5.13: Non-linear scaling of the hue similarity. The parameter is $s_{\max} = 100$.

[6]Since this parameter must be known in advance the actual computation has to be done in two steps, storing temporary results in between.

Applied to the colours of the centrifuge in figure 5.8 the hue similarity from equation 5.2 yields weights as in figure 5.14 (contrast-maximised to match the printable range of grey values).

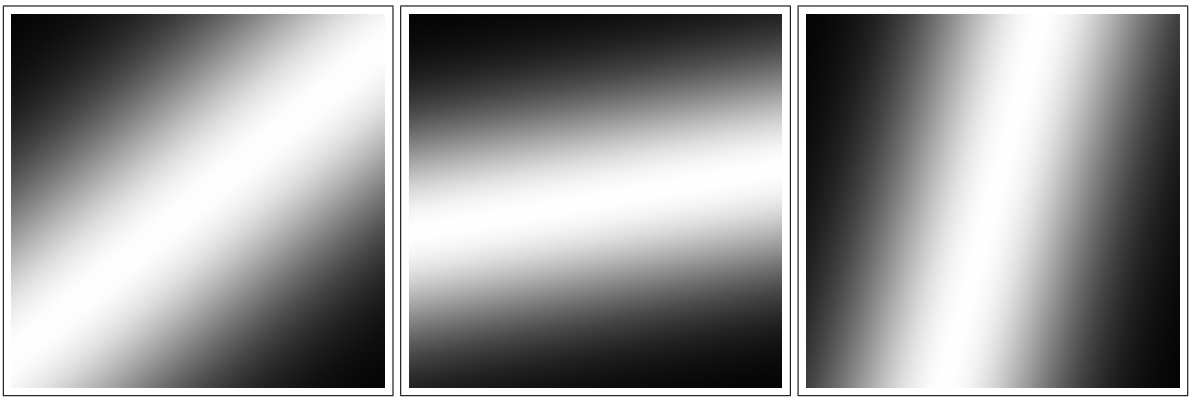


Figure 5.14: The hue similarity from equation 5.2 applied to the colours of the centrifuge.

Combined, the total similarity

$$d(\vec{c}, \hat{c}_\beta) \quad = \quad g\left(\vec{c} \cdot \hat{c}_\beta\right) \cdot h(|\vec{c} \cdot \hat{d}_\beta|) \tag{5.3}$$

applied to the colours of the centrifuge in figure 5.8 yields weights as in figure 5.15 (contrast-maximised to match the printable range of grey values). Applying these similarity weights to the centrifuge in figure 5.8 itself yields images of the similarity of regions to the colours as in figure 5.16.

These similarities can for visualisation purposes be used to reconstruct an image as in figure 5.17 of what this step of the vision process recognises as “coloured regions”. It is this image with all the non-coloured background removed that the following segmentation steps “see”, and not the original one.

5.2.2.1 Fixed Colours

This similarity measure is an intuitive and easy way to compare two colours, but the initial step of selecting the colour β to compare image colours to is yet unspecified. This step is important as it affects the parameters used in both the measure itself and the upcoming segmentation. Two different approaches have been researched.

The first approach is to use fixed colours. The centrifuge in figure 5.8 has regions with the colours blue (β = 40°), yellow (β = 190°) and red (β = 255°). The caps of the NUNC tubes are of a blue colour. The other objects are marked with labels which also consist of blue, yellow and red colours. These colours are strictly speaking not of 100% identical hue and saturation, but similar enough to motivate and allow a restriction to these three fixed colours. The advantage of this simplification is that the matching of colour hues can be reduced to the matching of two integer variables.

With this selection mode the sequence of the segmentation loop is slightly different from the one described earlier. The exact sequence of operation as in figure 5.18 is as follows:

1. One of the *predetermined* colours is selected,

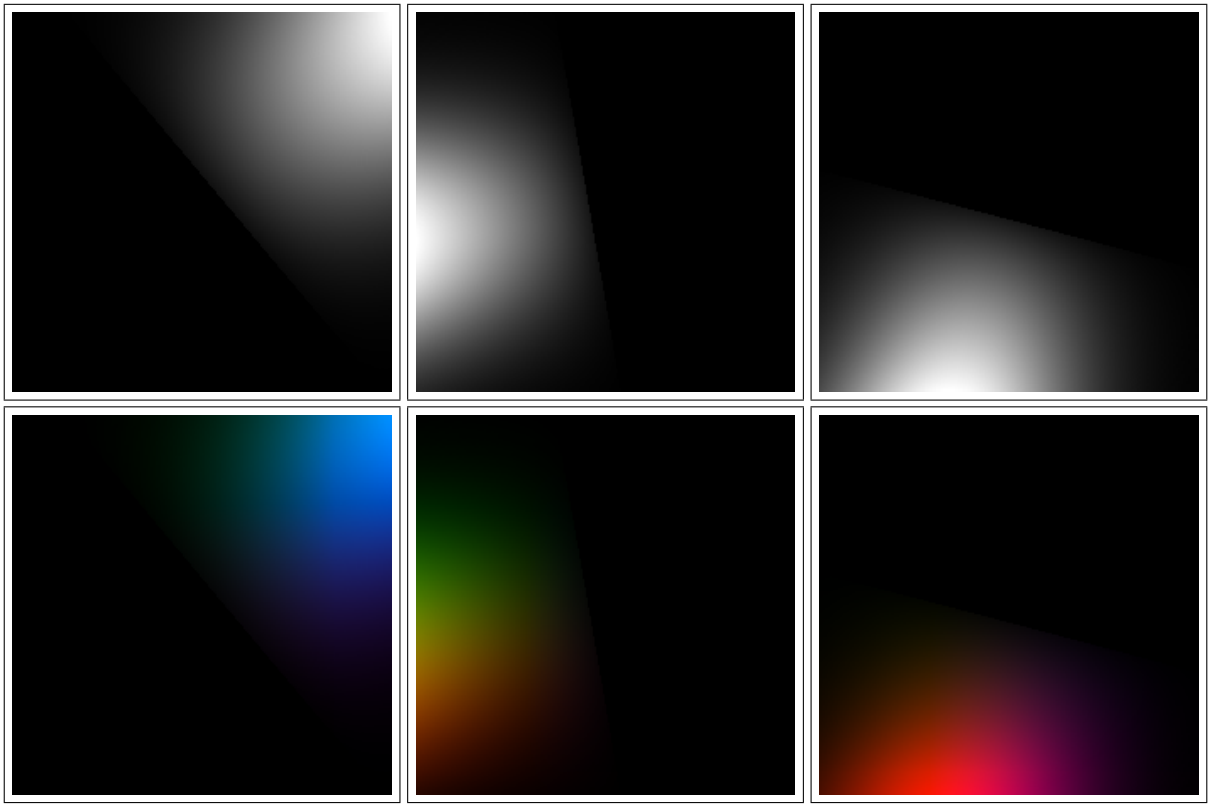


Figure 5.15: The combined saturation and hue similarity from equation 5.3 applied to the colours of the centrifuge. Upper row: original weights, lower row: weights applied to UV colour plane.

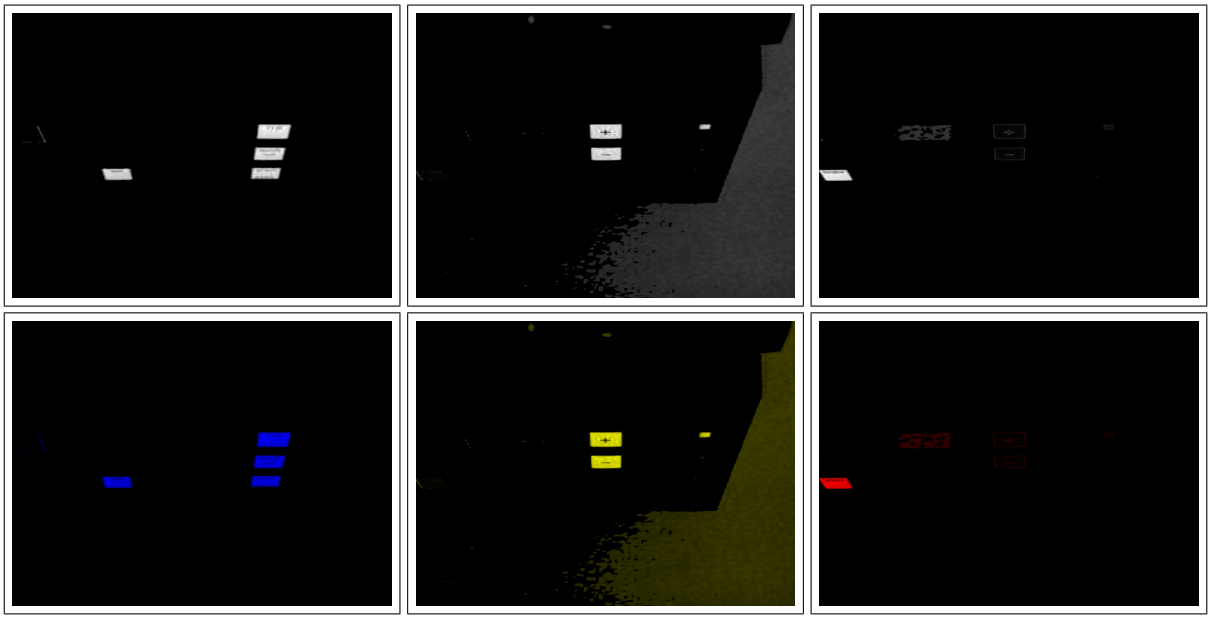


Figure 5.16: Similarity of regions of the centrifuge to colours of the centrifuge. Upper row: original weights, lower row: re-colourised weights.

Figure 5.17: Visualisation of the information reduction by focusing on coloured regions. Left: original image of the centrifuge, right: reconstruction based on the colour similarities in figure 5.16.

2. the similarity for this colour is computed,

3. the highest weighted pixel (seed) is determined and

4. a flood fill algorithm is used to grow a region around the seed.

This sequence is repeated from step 3 until no more pixel of sufficient similarity can be found. In this case the next colour is selected and the entire sequence restarted. If no more colours are left the entire sequence terminates.

Note that the regions are removed from the similarity image by the fill algorithm, but explicitly *not* from the original image. This is because the similarity measure allows colours with overlapping similarity to be mixed up, for example resulting in that a yellow region might be found as a red region (but with a poor similarity). If such an erroneously detected region would be removed it could no longer be found if the correct colour is selected. This leads to that some regions may be found more than once with a different colour.

This feature – though not being fatal – has to be considered a disadvantage of the approach. Another disadvantage is that the approach relies entirely on the a priori calibration. If this calibration is slightly wrong or if the hue of the illumination changes at run-time due to sunlight shining onto the scene this means that the measure will be slightly wrong too. As long as the hue shift is only small the measure will still weigh the “correct” colours highest, although with a loss of relevant bits. Nevertheless an alternative approach has been researched.

5.2.2.2 Adaptive Colours

The alternative approach is to select the colours adaptively based on the UV plane of each individual image. Such an approach requires a measure to base the selection on, and since the intention is to focus on coloured regions the colour saturation is the obvious candidate. Again, with this selection mode the sequence of the segmentation loop is slightly different from the one described in the beginning of this section. The exact sequence of operation as in figure 5.19 is as follows:

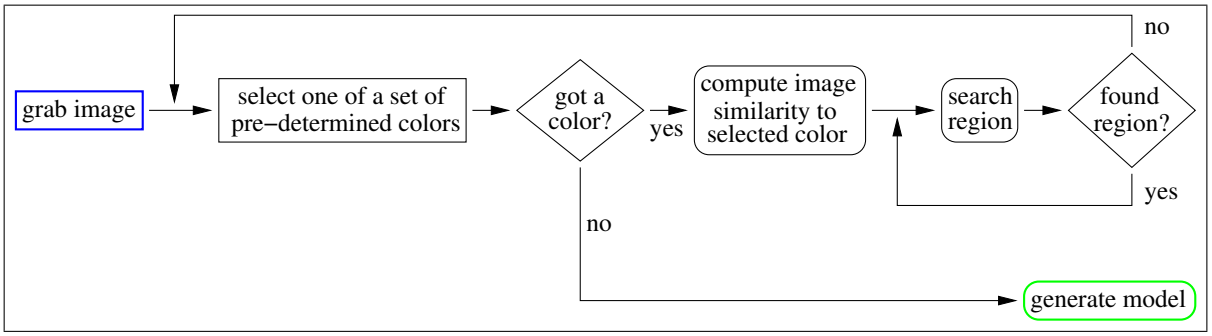


Figure 5.18: Execution flow of fixed colour selection.

1. Search the most saturated colour in the UV plane,
2. compute the similarity for this colour,
3. determine the highest weighted pixel (seed),
4. use a flood fill algorithm to grow a region around the seed and
5. remove the region from the original image.

This sequence is repeated from step 1 until no pixel of sufficient saturation can be found, in which case the sequence terminates.

This time the region has to be removed from the original image because of the search for the most saturated pixel: If it is not removed the same pixel is found again in the next run, resulting in an endless loop. It can be safely removed because due to the automatic colour selection the danger of mixing up colours under troublesome illumination conditions does no longer exist.

It has to be noted that the separation between the search for the most saturated pixel in step 1 and the search for the highest weighted pixel in step 3 means that it can not be guaranteed that these two pixel are the same. In practice this is very unlikely to happen because the highest saturated pixel can be expected to yield the highest weight. The only situation in which it might happen is when two spatially separated pixel have almost the same colour (hue *and* saturation) and due to rounding errors one of them is selected in step 1 and the other one in step 3. This is not a problem because since they belong to almost the same colour no error is introduced by mixing them up and the other one is very likely to be found in the next run.

The advantage of this adaptive approach is that it is more tolerant against suboptimal colour calibration and illumination. By removing regions from the image it avoids double recognition and therefore minimises the number of regions. Since the similarity weight can be assumed to be better suited for the coloured regions the thresholds used in finding that region can in theory be made more strict. If, however, they *are* made more strict this immediately means that the border of regions is not properly recognised as still belonging to the region. As a result the border is likely to be found as an additional region in a following run, leading to an increased number of total regions if the fragments are not merged again. In the direct comparison with the approach using fixed colour no immediate or significant advantage could therefore be found.

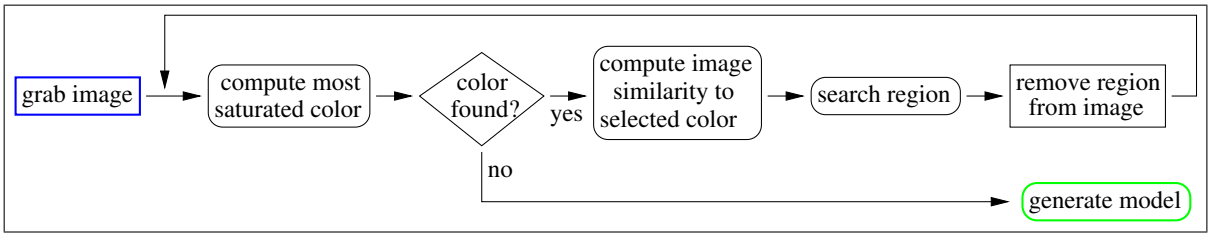


Figure 5.19: Execution flow of adaptive colour selection.

5.2.3 Flood Fill

After having selected the seed pixel as the highest weighted pixel using the above similarity measure the region corresponding to that pixel must be established. This is done by using a conventional *flood fill* algorithm on the similarity image. This recursive fill algorithm uses a four-neighbourship to grow a region. Each new pixel is checked against a threshold defining a minimum similarity and added to the region if it complies to the threshold. Due to the a-priori colour similarity computation this threshold is less critical to choose than in a multi-dimensional case, but still needs attention.

If the threshold is too low the region may not be completely filled, and if the threshold it too large the algorithm may "leak" out of the region and include parts of the background. In the former case the border around the region is likely to be accepted as a separate region later on, leading to a higher number of regions. This is not strictly speaking a problem, but leads to suboptimal performance. In the latter case, of course, the classification will fail.

Figure 5.20 shows a sample segmentation of a centrifuge button to illustrate this effect. The predetermined colour of the blue button has deliberately been mistaken as $\beta = 60°$ instead of the correct 40° to enhance the problem. Yet, even with this deliberate error the range over which the threshold has to be modified to trigger the problem is very large. Without the deliberate error the threshold is even less critical.

Figure 5.20: Influence of the threshold during a fill operation on a seed inside the upper right blue button. Thresholds used from left to right: 220, 128 and 20 from a range of [0..255]. Too large thresholds yield only a fragment of the button and too low thresholds leak into the background.

The result of the fill algorithm is a contiguous region of pixels around the seed, either as a mask image or as explicit coordinate list. Optionally, the hull of the region can be established by applying a simple criterion: Every pixel in the region whose 4 neighbours in the four-neighbourship are not all members of the region themselves belongs to the hull, and every

other pixel not. The hull can be useful for visualisation purposes like in figure 5.20, but is not required by the following processing steps.

Applied to the sample image of the centrifuge in figure 5.8 this combination of a colour similarity measure, seed selection and flood fill algorithm successively finds the coloured image regions. If no more coloured regions can be found the segmentation terminates. A visualisation of the process can be seen in figure 5.21.

The result of the entire segmentation process is a list of regions

$$R_i \quad = \quad (\beta_i, \Delta x_i, \Delta y_i)$$

where β is the colour hue of the region and Δx and Δy give the offset of the COG relative to the centre of the image in pixel. It is this list that all the following vision routines operate on, not the original images. The idea is that the reduction to the COGs still represents all the essential information needed to recognise the object and its position, as will be shown in the following section.

A particular advantage of the COG approach is that small segmentation errors at the border of regions do not overly affect the COG. For example, if from a centred 21×21 square region the upper leftmost pixel is not detected the COG changes from (0, 0) to (0.022727, 0.022727) only, and if the complete leftmost column of pixel is not detected it changes from (0, 0) to (0.5, 0) only. If the segmentation errors are symmetric they do not at all affect the COG. Because of this a region whose border has not been 100% accurately detected can still be used for classification. With a proper choice of the threshold the region list is therefore a very robust compressed representation of the original scene.

This can be seen in figure 5.22 where images have been artificially darkened and superimposed with noise to provoke suboptimal behaviour. Even though the segmentation of the regions becomes more and more disturbed with decreasing image quality the regions are still recognised. If only the COG of the regions is used for further processing the segmentation errors are unlikely to show up in later stages.

5.3 Object Recognition & Displacement Computation

The next step in the vision process is to use the extracted information to classify the objects and compute their position in the image. Classification basically means to take a set of features (patterns) belonging to several *classes* and to establish a function that allows to separate them[7]. This function can either be computed directly from the data (for example in polynomial classificators) or learnt iteratively (for example in *neural networks* (NNs)). All these approaches are based on the a-priori extraction of appropriate features, which is increasingly considered as inappropriate. The focus of research has in recent years moved to *appearance based approaches*, which take images as a whole and do not require feature extraction.

The objects used in this work are characterised by their coloured regions (labels, buttons and caps), which means that they consist of one or more simple geometric shapes. C. Berger has shown in [Berger 2000] that a *principal component analysis* (PCA) based approach is capable of classifying simple geometric shapes under some constraints. The most important constraint

[7] See [Niemann 1983], or chapter C.2.5 in the appendices.

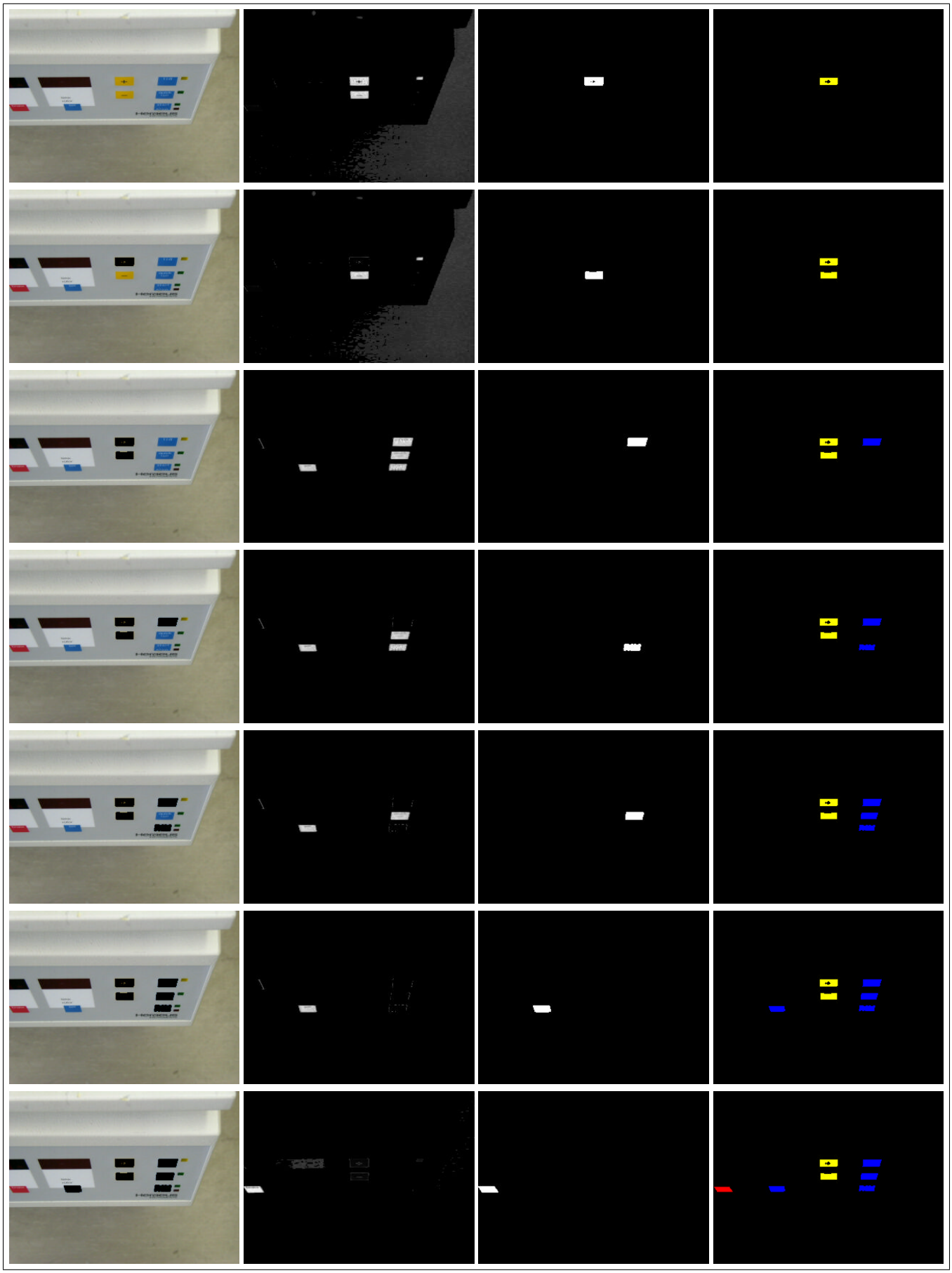


Figure 5.21: Successive segmentation of an image. The rows show the steps for each region from left to right: the original image, the colour similarity, the segmented region and the reconstructed image containing the regions found so far. The colours used from top to bottom are: yellow, yellow, blue, blue, blue, blue and red.

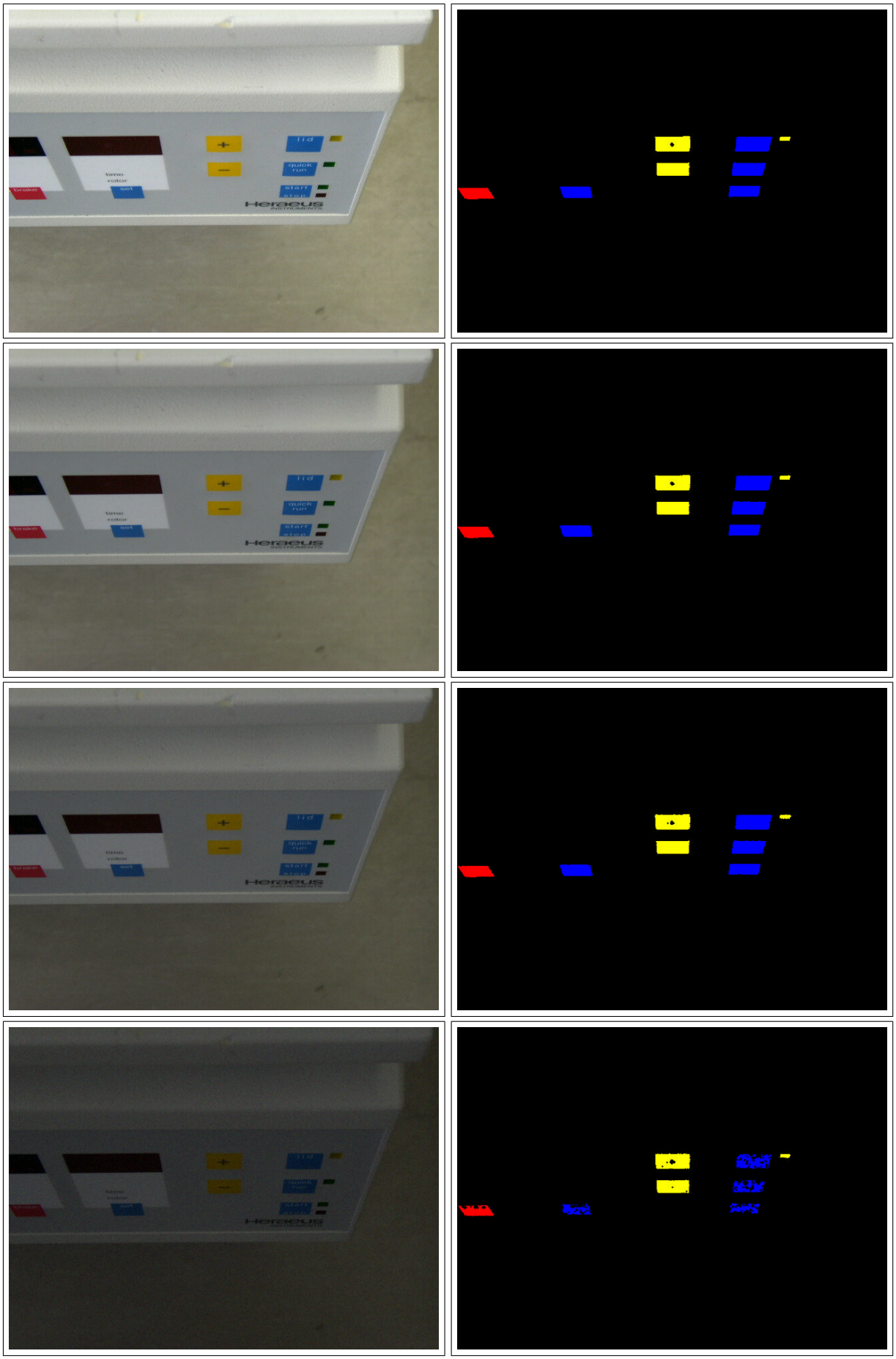


Figure 5.22: Segmenting an image of the centrifuge with different brightness and noise level. Each row shows the original image on the left and the reconstructed image with the segmented regions of the right. The rows from top to bottom show situations with 100, 77, 54 and 31% brightness and increasing noise level.

is that a set of images of the objects in all possible orientations must be given, because the PCA has only limited capabilities to interpolate between two situations. This is already a problem if the object is so large that it can not be rotated manually and that the robot holding the camera can not drive around it automatically.

A second drawback is that this approach only classifies the objects, but does not compute their position in the image. On the contrary, the PCA strictly requires a *normalised* image, that means a centred image of a normalised size and normalised brightness. This is because the PCA looks at differences in the image data without knowing whether the differences are relevant or not, so all irrelevant differences have to be removed in advance. In case of non-centred regions it can not be guaranteed that the PCA will indeed focus on the shape of the regions rather than its position.

A third drawback is that this approach is able to classify the shape of single regions, but not patterns consisting of several regions. Considering the problem of mixing up the object's appearance and position as relevant information a *bounding box* has to be placed around the regions belonging to the pattern in order to allow to normalise it in size and brightness. This, however, requires to know the position of the pattern in the first place – a typical chicken/egg problem.

A fourth drawback is that the behaviour of the PCA becomes entirely unpredictable when additional background regions not belonging to the object appear in the image. Basically, the PCA can only classify what it has learnt and is instantly confused when it hits upon a new situation. Since in the laboratory it can not be guaranteed that no background regions exist the PCA is therefore not suitable for this work. Tests have shown that this also applies to other appearance based approaches like *output related features* (ORFs). An entirely different solution is therefore needed.

The approach proposed in this work actually makes active use of the distinction between single regions (simple objects) and patterns of several regions (complex objects). The only points where the recognition of simple objects is needed are the rings used in detecting NUNC/Cedex tubes or slots in their storage racks. Since there is no point in trying to find a tube or slot in a storage rack if the position of the rack itself is unknown a cascaded approach is suggested: First find the object (*outer classification*), then find the rendezvous point(s) on the object (*inner classification*). This cascading allows certain constraints about the situation during inner classification, like only minor displacements caused by inaccuracies of the outer classification will occur and the situation can otherwise be assumed inoffensive[8]. From these constraints simplifications of the classification itself can in turns be drawn. Using these simplifications, the two parts of the cascaded classification can be given as follows:

Inner classification covers the detection of simple object consisting of only one region (tube caps or rings used around slots of storage racks). These objects are actually not classified at all. Instead, only the segmentation step is applied and centre-most region is then

[8]The main purpose of the vision system is to deal with a possible confusion of devices and/or larger displacement of devices. Both problems are caused by handling errors by human personnel, but once the correctness of a device has been assured certain assumptions about it have to be made, like the cage of the centrifuge is not modified and the positions of the slots in a rack are not changed. Due to limited sensoric capabilities not every imaginable deviation from the desired state can be detected, and therefore some constraints must be allowed.

assumed to be the tube/ring[9]. Since the outer classification of the corresponding storage rack has already asserted certain constraints this simplification still provides sufficient information to safely grasp/place a tube. The inner classification only needs the image segmentation part described in section 5.2 and is not further described here.

Outer Classification covers the detection of complex objects consisting of more than one region (all other objects like the storage racks themselves, the centrifuge etc.). These objects require a true classification because no assumptions about the correctness of a device or its displacement can be made. Since PCA/ORF based approaches have proven to be not applicable a model based approach as in figure 5.23 has been established instead. This approach needs only a single image of the desired situation for training, can be easily extended by additional objects and yields the classification and position computation in a single step.

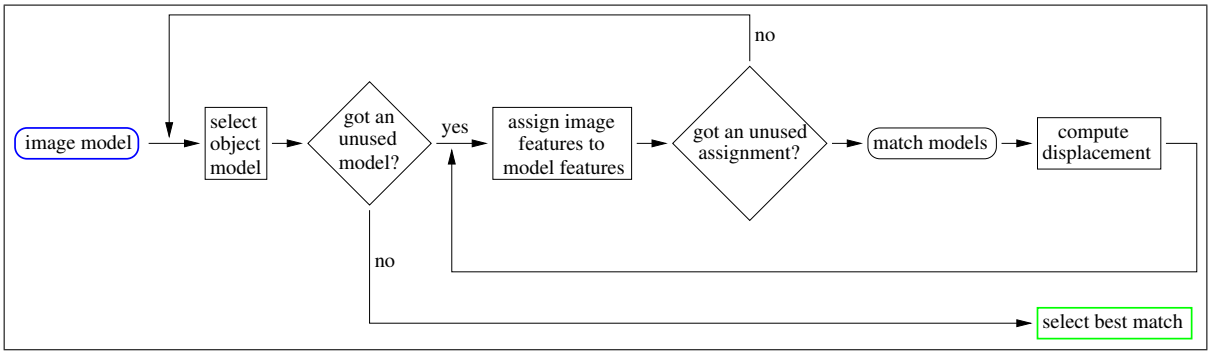


Figure 5.23: Execution flow at object recognition (outer classification).

5.3.1 Object Models

The idea behind using a model is to not rely on an unknown internal representation of a situation in a PCA or NN, but to have more control over the classification process. The obvious way to achieve that is of course to explicitly compute features from the object. When using only a single feature it must be simple enough to be reliably detected and yet complex enough to allow to reliably distinguish between multiple objects. Since Berger's results do not make this appear too promising the alternative approach of using a set of simple features that can be reliably computed and use their spatial relationship to distinguish between objects has been chosen. The detection of an object is therefore reduced to the detection of a set of features – the model.

This leaves the detection of the individual features, which is not trivial. Since the purpose of the vision system is not to classify a normalised image of the object but to allow certain degrees of freedom (translation and rotation), the detection of the features must be invariant with respect to these degrees of freedom. For example, if a classifier is able to detect triangles that are standing upright it can only detect translated triangles, but not rotated ones. The only shape that is fully translationally and rotationally symmetric is a circle, hence the COGs from the region list are used as features.

[9]The question of whether the "centre-most" region is really the desired tube/ring can be answered by looking at its distance from the centre. If that distance is higher than some threshold the slot must have been empty and a neighbouring tube has been selected.

From this set of 2-dimensional points $\vec{p}_i$ a subset is selected according to some constraints as the model M of the object, which can finally be given as:

$$M = (\vec{p}_1, \vec{p}_2, ..., \vec{p}_n)$$

The selection process has to be done manually because the model has to satisfy some constraints which can not trivially be established automatically. One constraint is that it has to be ensured that only relevant features are selected, another one is that the selected features must be reliably detectable under all possible translations and rotations and, finally, the feature sets must be sufficiently different for different objects to allow a reliable classification. A visualisation of the model generation steps so far can be seen in figure 5.24.

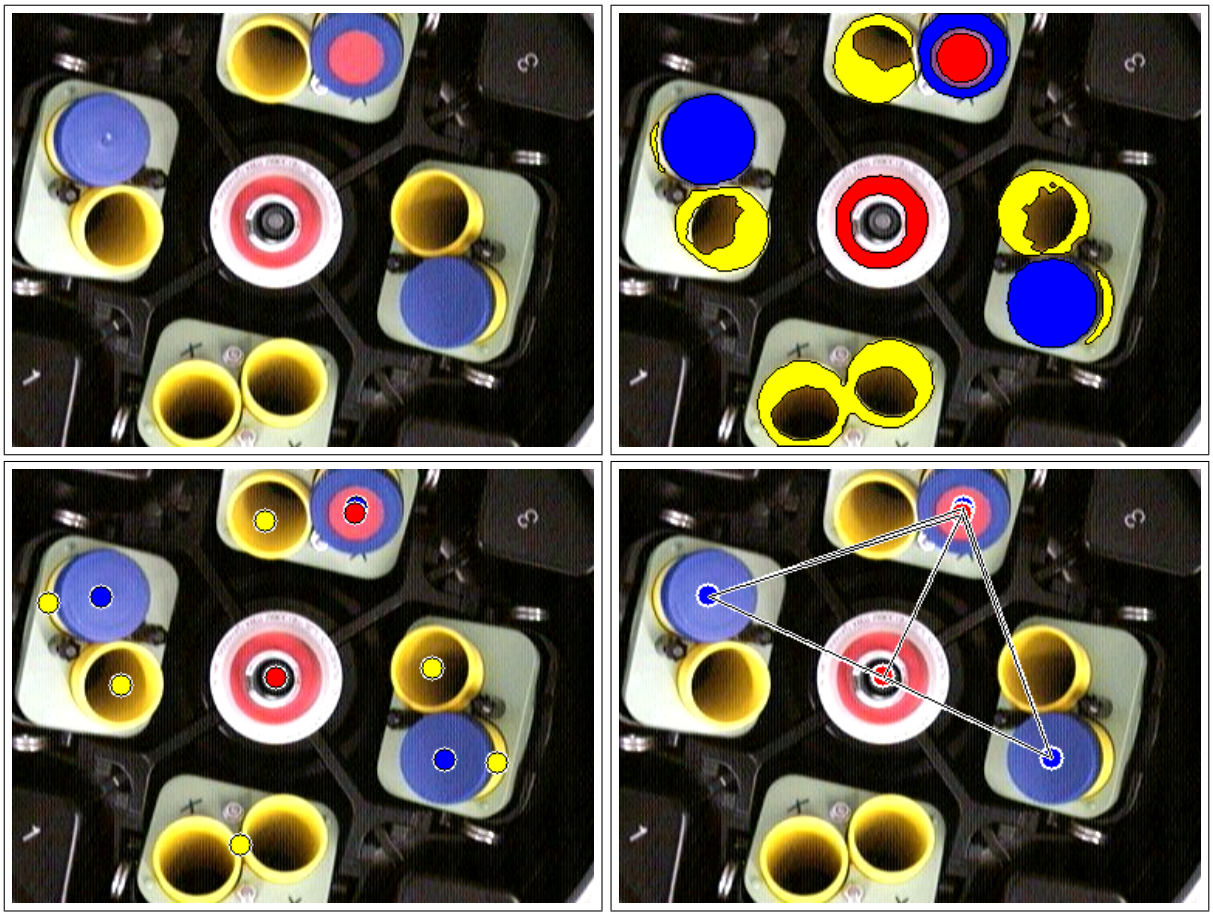


Figure 5.24: The separate steps during model generation: The original image (upper left), the segmented regions (upper right), the COGs of the regions (lower left) and the model as a subset of the COGs (lower right).

The constraint that the models have to be sufficiently different for different objects is important because it affects the classification performance more subtly than the others. Speaking in terms of separation by a hyper-curve, if the patterns are too close together the question of whether they can still be separated becomes a question of measurement noise. For example, in the set of three models with three one-dimensional features

$$\begin{aligned} M_1 &= (-1, 0, 1) \\ M_2 &= (-1.95, -0.98, -0.03) \\ M_3 &= (-1, 0, 2) \end{aligned}$$

M_1 and M_2 are quite similar because M_2 is just a translation of M_1 plus some noise. The difference between any of them and M_3 is much higher. The feature set would allow to confuse objects 1 and 2 very easily and does therefore not yield good models. This issue has to be addressed at feature selection, but does not pose a problem if the models can be explicitly designed.

Except from the centrifuge all devices are labelled with special markers as in figure 5.25 that satisfy the above constraints. The markers can be printed with an ordinary colour laser printer and attached to the devices. They implement a 4-bit code to distinguish 16 different devices[10], yet they are unique unter all translations and rotations (but not mirrorings).

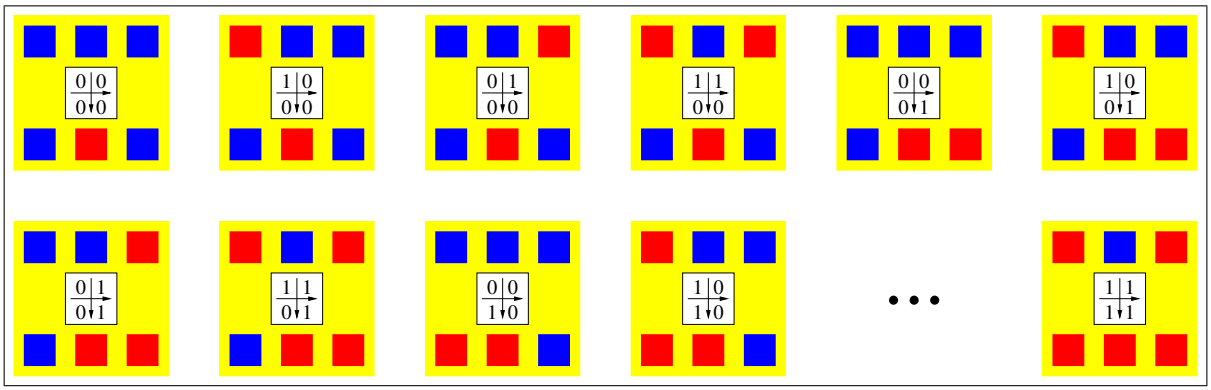


Figure 5.25: The markers attached to the devices.

The result of these considerations is a generic model of an object. The model consists of a list of xy-COGs of model regions and their colour. The COGs are two-dimensional coordinates because the system only has a single camera and therefore can not detect depth information. A stereo camera mechanism required to do this would be very difficult to attach to the tool while keeping the tool compact and the system would not necessarily benefit from it. Therefore only 2d-models are used in the following steps.

5.3.2 Model Generation

Model generation is done off-line in a learning phase before using the system. With PCA or NNs "learning" means to iteratively present the system samples and the desired output. The number of learning data is usually very high because all possible views or variations of all possible objects have to be considered. During the iteration the system then learns to minimise the output error according to a learn rule and thus to classify the objects. Since the effort to do this scales linearly to the number of learn data and the number of learn iterations this process can take quite some time.

Contrary to these approaches the established system has implicit knowledge (another model) about how the appearance of the object changes under translation and rotation. It it explicitly designed to use this knowledge to compute this displacement from a given image without having stored a dedicated sample for each displacement. The only sample the system needs is one of the object it its *desired pose*. For this an integrated tool exists with which the administrator can

[10] For the current laboratory setup this is more than enough, yet – if it should ever become insufficient – a code with more than 4 bits can be easily designed.

- move the robot (camera) over the object in the desired pose,
- take an image,
- segment the image into a list of coloured regions (or rather their COGs only),
- build a model by selecting a proper subset of the COGs and
- store the model.

Requiring only a single model per object for all possible translations and rotations is a great advantage. It allows to handle devices where it is not possible to automatically generate an extended set of learn samples (e.g. the centrifuge). Furthermore it reduces the learning effort by requiring only constant resources (leaning time and storage space).

Another advantage is that it is easily possible to add new objects at run time. With PCA or NNs it is not possible to add more objects after the learning step because only abstract information about the objects is stored and this information is spread over the entire storage space. When a NN is given a new situation it will learn information about it, but at the price of destroying information about the old ones. The only way to really add new situations is to keep the old set of samples, add the new one and re-learn the entire set. This is of course not desirable because it requires a huge amount of resources. With the established system adding a new object is reduced to adding a new model. It does not affect the knowledge about the old objects and the effort is again constant.

5.3.3 Model Matching

With having only a primitive model of the desired pose of an object the classification and computation of the displacement is necessarily somewhat more complex than with a NN or PCA. The approach used in this work is based on *matching* models, which means that a model is generated automatically from a camera image of a scene and then compared to the manually generated models describing the desired poses of the known objects. In a single step this yields both the displacement of the camera model as well as the correct object class in the first place.

5.3.3.1 Displacement Model

Given a model $O = (\vec{o}_1, \vec{o}_2, ..., \vec{o}_n)$ of an object in a known situation and a model $C = (\vec{c}_1, \vec{c}_2, ..., \vec{c}_n)$ of a camera image of an unknown situation a linear displacement D is assumed between each corresponding feature according to

$$\vec{c}_i = D \cdot \vec{o}_i \tag{5.4}$$

Since the features are 2-dimensional only translations along the x- and y-axis and rotations about the z-axis apply and the displacement – written as a 4×4 homogenous matrix – can be given as

$$D = \begin{bmatrix} \cos(\varphi_z) & -\sin(\varphi_z) & 0 & d_x \\ \sin(\varphi_z) & \cos(\varphi_z) & 0 & d_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inserted to equation 5.4 the equation expands to a *linear equation system* (LES)

$$\begin{pmatrix} c_{i,x} \\ c_{i,y} \\ 0 \\ 1 \end{pmatrix} = \begin{bmatrix} \cos(\varphi_z) & -\sin(\varphi_z) & 0 & d_x \\ \sin(\varphi_z) & \cos(\varphi_z) & 0 & d_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} o_{i,x} \\ o_{i,y} \\ 0 \\ 1 \end{pmatrix}$$

which yields two separate equations

$$\begin{aligned} c_{i,x} &= \cos(\varphi_z) \cdot o_{i,x} - \sin(\varphi_z) \cdot o_{i,y} + d_x \\ c_{i,y} &= \sin(\varphi_z) \cdot o_{i,x} + \cos(\varphi_z) \cdot o_{i,y} + d_y \end{aligned}$$

with – taking $\sin$ and $\cos$ as two independent variables – four unknowns. To indicate this, the equations can be rewritten as

$$\underbrace{\begin{pmatrix} c_{i,x} \\ c_{i,y} \end{pmatrix}}_{\vec{c}'} = \underbrace{\begin{bmatrix} -o_{i,y} & o_{i,x} & 1 & 0 \\ o_{i,y} & o_{i,x} & 0 & 1 \end{bmatrix}}_{K'} \cdot \begin{pmatrix} \sin(\varphi_z) \\ \cos(\varphi_z) \\ d_x \\ d_y \end{pmatrix}$$

with $K' \in \mathbf{R}^{2\times 4}$ holding the "new" matrix of weights. This LES can not be solved for the unknowns because it is under-determined. Since, however, the same displacement D applies to all features of a model (which is why there is only a D, and not several D_i) multiple equations from multiple features can be joined into a larger LES. Using $2N$ equations from $N > 1$ features it can be rewritten as

$$\underbrace{\begin{pmatrix} c_{1,x} \\ c_{1,y} \\ c_{2,x} \\ c_{2,y} \\ \vdots \\ c_{N,x} \\ c_{N,y} \end{pmatrix}}_{\vec{c}} = \underbrace{\begin{bmatrix} -o_{1,y} & o_{1,x} & 1 & 0 \\ o_{1,y} & o_{1,x} & 0 & 1 \\ -o_{2,y} & o_{2,x} & 1 & 0 \\ o_{2,y} & o_{2,x} & 0 & 1 \\ \vdots & & & \\ -o_{N,y} & o_{N,x} & 1 & 0 \\ o_{N,y} & o_{N,x} & 0 & 1 \end{bmatrix}}_{K} \cdot \begin{pmatrix} \sin(\varphi_z) \\ \cos(\varphi_z) \\ d_x \\ d_y \end{pmatrix} \tag{5.5}$$

with $K \in \mathbf{R}^{2N\times 4}$.

5.3.3.2 Least-Squares Fit

With $N = 2$ features equation 5.5 could be solved directly, but would not yield a good recognition. If the computation of the displacement depends on the location of only two points inaccuracies in detecting have a too large influence on the result. Furthermore, if the colour code consists of only two points it would not allow to distinguish two objects, because two points are already needed to distinguish all possible rotations of one object. Classification and robustness against detection inaccuracies can only be achieved by using more than two features.

Unfortunately, with $N > 2$ features equation 5.5 can again not be solved directly because it is over-determined. K is non-square and can therefore not be inverted. The situation is identical with the problem in section 3.2.4, and the solution is again to use a *least-squares fit* (LSF).

Instead of inverting K directly the the *pseudo inverse* (PI)

$$K^+ \;=\; (K^T K)^{-1} K^T$$

as an approximation $K^+ \approx K^{-1}$ of the real inverse has to be used[11]. The unknowns can then be computed as

$$\begin{pmatrix} \sin(\varphi_z) \\ \cos(\varphi_z) \\ d_x \\ d_y \end{pmatrix} \;=\; K^+ \cdot \vec{c}$$

One way to check for consistency would be to ensure that

$$\arcsin(\sin(\varphi_z)) \;=\; \arccos(\cos(\varphi_z)) \;=\; \varphi_z$$

plus or minus some quadrant corrections, but this is not robust enough. Instead, by inserting the unknowns in the original LES from equation 5.5 the *mean-square error* (MSE)

$$e_{C,O} \;=\; \left|\left| \vec{c} - K \cdot \begin{pmatrix} \sin(\varphi_z) \\ \cos(\varphi_z) \\ d_x \\ d_y \end{pmatrix} \right|\right|$$

from the LSF can be computed. This error $e_{C,O}$ expresses the average distance between features from the camera model C and the object model O and is expected to be low for correct matches. By computing the displacement and the associated error for all stored models and choosing the one with the smallest error the camera image can therefore be classified and the displacement obtained in a single step.

The MSE $e_{C,O}$ has to catch any aspect that is not part of the displacement, like lens or perspective errors. A lot of different lens errors exist, but for the purpose of this work only the radial lens error is important. Radial lens error means that a ray that is coming through the outer parts of a lens is projected differently compared to a ray that is coming through along the optical axis: The focal point of a lens depends on the radial distance from the optical axis through which a ray comes. This can be compensated by applying a radial stretching of the image as in figure 5.26, but this correction is not utilised in this work. The reason for this is that the error introduced by omitting the correction is below the threshold that is required to mix up two models, as will be shown in section 5.5.

The perspective error is raised by the fact that the vision system uses only two dimensions. For feature patterns that lie in a plane that is parallel to the image plane only the radial lens errors apply because the entire situation is two-dimensional. If the planes are not parallel – as in case of the centrifuge – perspective distortion occurs if the pattern is looked at from different view

[11] See [Moore 1920] and [Penrose 1955].

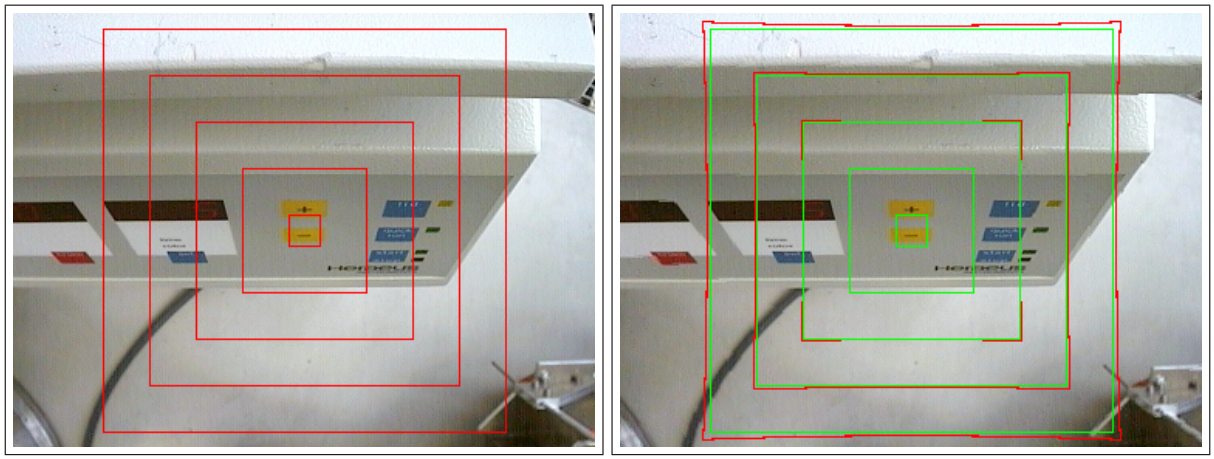


Figure 5.26: Radial lens error: Original (left) and corrected image (right).

points. The spatial relation of features changes with the view point, for example leading to that something that is “above” something else in the model will additionally shift a bit sideways. Figure 5.27 shows an illustration of this perspective effect. This of course affects the model matching because the two patterns no longer fit exactly, but again the error introduced by this is below the threshold that is required to mix up two models, as will also be shown in section 5.5.

Figure 5.27: Perspective effects on a titled pattern: The spatial relation of features changes with the view point.

5.3.3.3 Permutations

One issue and potential error that is so far uncovered is raised by the fact that in the introduction it was said that the displacement D is assumed between each *corresponding feature*. This means that features from the two models C and O have to be mapped in advance, which can become problematic in several ways:

- If the model C contains more features than O the mapping is possible if the subset of those f_C features corresponding to f_O features can be found. However, even if C contains more features than O this does not necessarily mean that it contains the *same* features as O. It may happen that one or more of the f_O features are not detected, but features resulting from the background and/or other objects are detected instead. Therefore the existence of the mapping can not be guaranteed.

- If the model C contains the same number of features as O this still does not mean that the mapping can be established, because of the same reasons as above. Therefore the existence of the mapping can again not be guaranteed.

- If the model C contains less features than O the mapping is definitely not possible.

If the mapping can not be established this means the models can not be matched using the approach presented here, no classification can be obtained and no displacement be computed. One possible attempt is to try and take away features from O and repeat the mapping attempt, but since this – if it works – leads to reduced reliability it has not been included in this work. Instead the program will try and repeat the entire recognition step with a new image.

If this continues to fail for example because the light is switched off at night it remains stuck in an endless loop, optionally notifying the operator. This is considered more safe than continuing with uncertain results.

But even if the mapping can be established because all features from O are contained in C – and let for simplicity be assumed that exactly only the features form O are contained in C – it has to be actually found. Since the features are detected according to a complex sequence by decreasing colour similarity (which varies from image to image) and from the bottom right to the upper left image corner (which, since the orientation is yet unknown, does not allow to identify *specific* features of a model) the plain order of detection does not permit such a mapping. With – contrary to the localisation of the mobile platform in section 3.2 – no initial guess about the displacement that allows to identify corresponding features except from the colour (which is the same for several, but not all features) there is no way but to try all possible permutations.

The possible number of permutations is given by the *binomial coefficient* $\binom{c}{m}$ for drawing m objects from a pool of c indistinguishable objects, multiplied by $m!$ because in the case of mapping features the objects are not indistinguishable and the order in which they are drawn does matter. This makes up for a total of

$$\#\text{perms} \;=\; \frac{c!}{(c-m)!}$$

permutations, which leads to a triangular structure that – starting at $c = 1$ – begins as

1
2 2
3 6 6
4 12 24 24
5 20 60 120 120
6 30 120 360 720 720
7 42 210 840 2520 5040 5040
8 56 336 ...

In this triangle the last two numbers of each row are the same because if every features is assigned except the last one, then there is exactly one possibility left to assign that feature to.

With the $m = 7$ features as used by the markers in figure 5.25 this means that in order to recognise the marker $7! = 5040$ permutations of assignments (mappings) have to be passed through the least-squares fit[12]. Since this step has to be repeated for all N objects to be recognised $N \cdot 5040$ (80640 if all 16 markers from figure 5.25 are used) of the least-square fits have to be done. The number of permutations scales over-exponentially, which is not a good behaviour. Approaches to reduce the dimensionality of this problem are discussed in subsection 5.5.3, but have not been incorporated in this work.

Apart from these resource requirements the least-squares model matching stands out because of its absolute robustness. If all necessary features are found the MSE $e_{C,O}$ allows for a safe detection of the correct object and displacement, and if not all necessary features are found it also allows to safely detect that. Results from experiments demonstrating this are given in subsection 5.5.

5.4 Iterative Displacement Compensation

Using the above model matching an object can be classified and its displacement compared to a desired pose can be computed – in image coordinates. In order to compensate for this displacement it needs to be converted into arm coordinates, taking into account the way the camera is mounted to the tool[13]. The pose of the camera coordinate system relative to the arm coordinate system is established using an off-line calibration, but the translational scaling is not. As a result, the control program knows in which direction to move the arm if the vision says that the object is "to the left", but it does not know how far to move.

The reason for why the translational scaling – the millimetres per pixel – is not established off-line too is that it is different for some of the objects. It depends on the height with which the camera is held over the object: If the height is small it will be small too because the image becomes larger, and if the height is large it will be large too because the image becomes smaller[14]. Since with a monocular and two-dimensional vision system the height can not be measured automatically the translational scaling can not be established automatically.

[12] And that only applies if $c = 7$ because no additional background features are found. If they are ($c > 7$), this number increases even more.

[13] In this context "arm coordinates" stands for what is known as the T_6 frame in RCCL and many other works. The tool itself is often assigned a T_{TOOL} transform that, if applied to T_6, leads to the T_{TCP} coordinate system (the *tool centre point*). In terms of kinematics the camera is just a different "tool" leading to a different "tool centre point".

[14] The rotational scaling is not affected by this because the angle stays the same no matter from what distance it is observed.

One theoretical possibility to avoid this problem is to use additional knowledge about the models. If the z-distances to the features in the model were known to the system it could establish the scaling automatically. This, however, introduces an additional situation where upon adding of a new object manual measuring is required rather than automatic learning, which is not desirable. Furthermore, it works only for the classification of complex object models and not single regions as used in centring over tubes and slots.

Instead of calibrating the correct scaling off-line an uncalibrated approach as in figure 5.28 is used, using the minimal possible value of the scaling for the closest possible object. 90% of the displacement as computed by the scaling are compensated and the process is repeated until the displacement is small enough. The resulting iterative process usually requires more just one run to centre over an object, but prevents overshoots and is therefore stable.

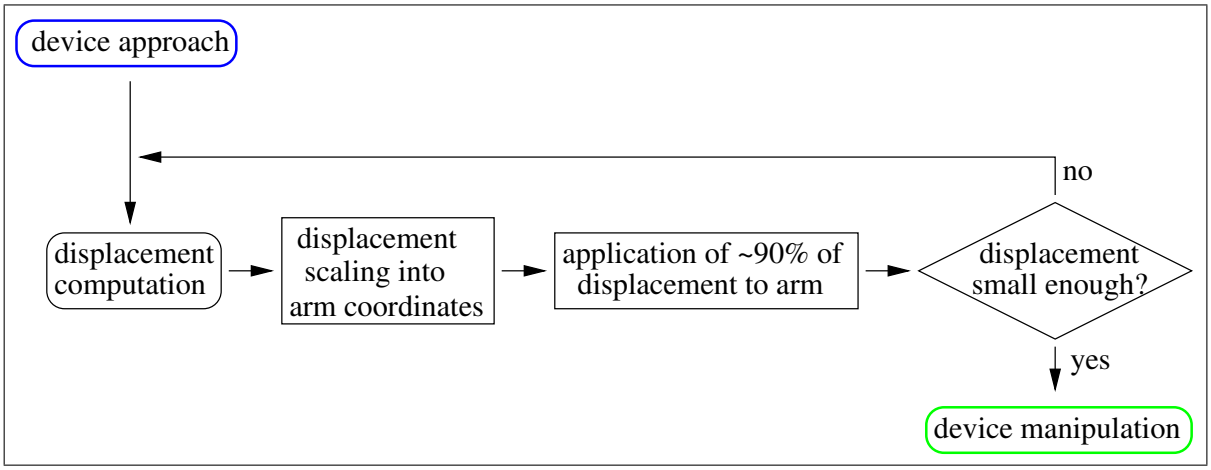


Figure 5.28: Execution flow at displacement compensation.

Figure 5.29 shows an excerpt of a log of the detected positional and orientational errors during a test run involving several objects. The MSE $e_{C,O}$ is given as a measure of the quality of the classification. The loops for the different objects can be clearly seen from the decreasing positional error (in pixel) during displacement compensation. In almost all cases the positional error can be brought down to one pixel only. The orientational error (in degree) is much smaller than the positional error because it is far less affected by the platform motions.

For the last object (the centrifuge) the MSE also decreases from an initially significantly higher value compared to the previous objects, demonstrating the perspective error mentioned above and showing that it does not disturb the classification and displacement compensation.

5.5 Experiments & Results

Being the most important part of the arm control software the vision system has been tested in several experiments, which are presented and discussed in subsection 5.5.1. In Subsection 5.5.2 the vision system and the results of the experiments are summarised, while subsection 5.5.3 gives a brief outlook in the possible future.

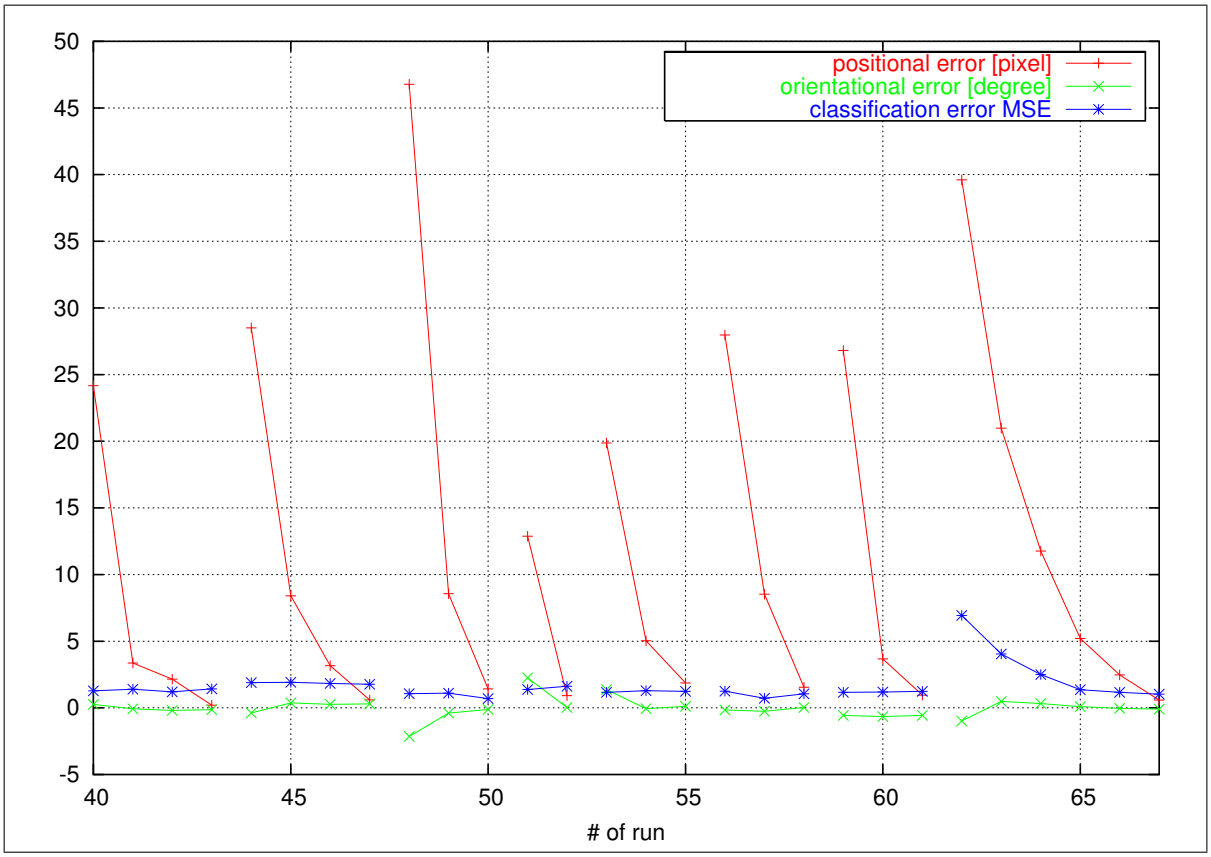


Figure 5.29: The positional and orientational errors iteratively decrease during the displacement compensation. The figure shows results for several different objects during a test run. In almost all cases the positional error can be brought down to one pixel only.

5.5.1 Experiments

The intended goal of the vision system was to be as tolerant against changing environment parameters as possible in general, and not requiring a standardised illumination in particular. Allowing a wide range of input parameters means that mandatory processing parameters and thresholds may be difficult to find, and yet a typical vision system requires a lot of these. The idea behind the established system was to keep these thresholds as non-critical as possible by preferring meaningful operations and parameters over abstract ones – for example the fill threshold on a colour similarity “image” is not just an abstract entity, but does have a descriptive meaning.

To verify that these goal are met several experiments have been done. These experiments cover:

- The basic repeat accuracy,
- the sensitivity against the robot arm positioning precision,
- the sensitivity against the most important vision parameters,
- the sensitivity against lens/perspective errors
  - during translation of a planar marker,

  - during translation of a non-planar (titled) marker and
  - during translation of the centrifuge, and

- the suitability of the the mean-square error (MSE) $e_{C,M}$ for classification.

The experiments have been repeated on three different objects:

- Experiment "20031201-planar" uses a planar marker. It is primarily affected by lens errors only and should otherwise be the best case.
- Experiment "20031201-tilted" uses a tilted marker. This introduces perspective errors and therefore larger errors, but since the marker is still rather small the perspective errors are expected to be small too.
- Experiment "20031210-centrifuge" uses the front plate of the centrifuge. Compared to the titled marker the front plate of the centrifuge is titled even more and the pattern therefore more difficult to detect[15]. Also, the pattern is larger and therefore more subject to perspective errors. It is therefore expected that the centrifuge presents the worst case.

5.5.1.1 Basic Repeat Accuracy

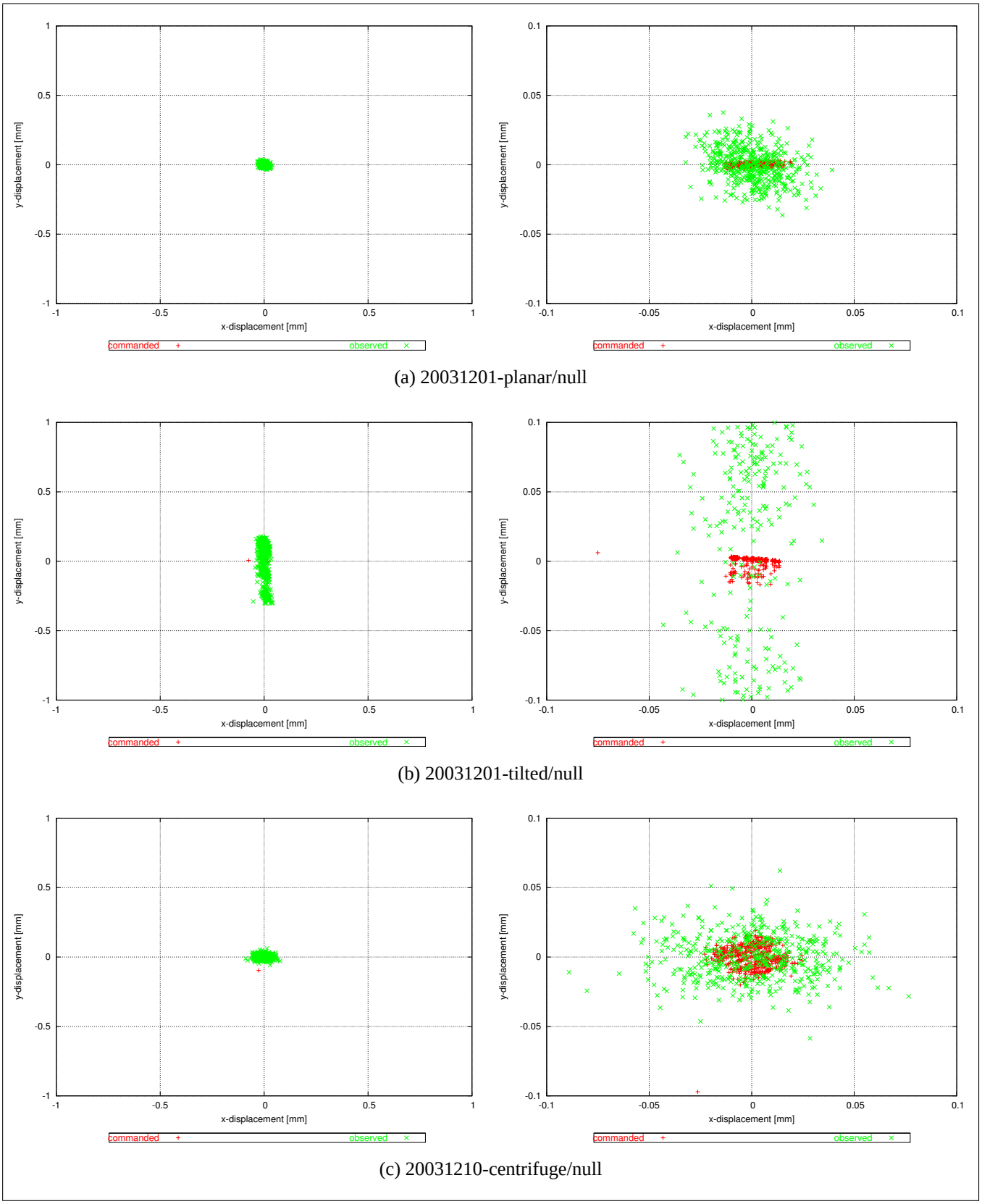
The basic repeat accuracy is defined to be the noise of the vision output when the input is as constant as possible. Several experiments have been done in which the camera is centred over an object, but the arm is not moved during the experiment (a "null" motion is applied) and the vision parameters are also kept constant. In such an experiment the segmentation of the regions will change only minimally due to imaging noise and rounding effects, and the purpose of the experiment is to establish the order of magnitude these effects have on the output.

This and the following experiments will produce separate, but not independent measurements for the arm and the vision. The arm measurements give the readings from the joint position encoders translated by the forward kinematic. They are labelled "commanded" and given in millimetres natively. The vision measurements give the output from the vision system that is relative to the arm. They are labelled "observed" and are natively computed in pixel, but have been converted to millimetres with a manually established scaling factor that is given in each experiment. The scaling factors are only used for the purpose of comparison in the experiments, but not during the compensation.

As can be seen in figure 5.30 the noise of the arm measurement during null motions is – as expected – not zero. This is consistent with the fact that in the experiment it could be observed that the arm was slightly vibrating. With no brakes applied the arm control is continuously evaluating its control loop to keep the arm at a steady position. Since the PID controller used for this requires a positioning error to generate commands and since the gravity pull the arm out of its desired position it can not stay exactly at that position without any noise. Since the camera is mounted to the arm the vision error therefore can not be smaller than the arm error – even with a perfect vision system.

The standard deviations of the distributions given in table 5.1 show that the basic noise is very low. The median order of magnitude of the vision error is between 1 and 2/100 millimetre. The y-axis of experiment "20031201-tilted/null" in subfigure 5.30(b) with a standard deviation of

[15] It becomes invisible at comparatively small translations due to the large tilt angle.



(a) 20031201-planar/null

(b) 20031201-tilted/null

(c) 20031210-centrifuge/null

Figure 5.30: Scatter plots demonstrating the basic repeat accuracy during null motions. The rows show the experiment on three different objects: A planar marker, a tilted marker and the centrifuge (which has a tilted front plate). The left column shows the original plots and the right column closeups.

1/10 millimetre is an outlier. The value is in the same range as in the experiments with dummy motions in figure 5.31(c) and therefore hints towards external disturbances of a similar kind. In the experiment the robot has been standing in a large room whose floor was slightly shaking because of heavy objects being moved in adjacent rooms, thus causing the arm to swing a little bit.

Experiment		σ_x	σ_y	Scaling
20031201-planar	commanded	0.007741	0.001238	–
	observed	0.012606	0.012835	0.39
20031201-tilted	commanded	0.007496	0.003776	–
	observed	0.014794	0.133436	0.40
20031210-centrifuge	commanded	0.009487	0.008524	–
	observed	0.024882	0.016175	0.53

Table 5.1: Standard deviations σ_x and σ_y for the experiments in figure 5.30. All σ values are in millimetres, the scaling factors in millimetres per pixel.

Overall, the order of magnitude of the error is roughly the same for all experiments. The error is very small because the segmentation is very robust. As long as a segmented region only changes by a few pixels at its hull this does not affect the centre of gravity (COG) to a noteworthy degree, and if it does this effect is still levelled by the least-square fit (LSF) which mingles all COGs. The overall resulting noise is therefore fully neglectable.

5.5.1.2 Robot Arm Positioning Noise

In another experiment the robot is commanded to perform "dummy" motions, that means motions which always bring it back to the initial position. This experiment can be used to judge the influence of the robot arm positioning on the vision, because due to the limited accuracy of the arm's sensors and actuators it will not return to the exact same position. In terms of centring above an object this means that even if the displacement was computed absolutely correct the motion compensating it will still leave a small error. The purpose of this experiment is to establish the order of magnitude of this error and the effects this has on the vision accuracy.

The amount of the positioning error that affects the depends on several kinematic aspects. For example, if the arm is completely outstretched a rotational error of 1 unit in the shoulder leads to a larger displacement than if the arm is in hinged position. Likewise, some joints affect the cartesian position more than others, and so the error depends on the current pose of the arm. The experiment has therefore been repeated with two different motions:

1. Experiments denoted with "dummy1" use a constant motion of 50 mm along the z-axis perpendicular to the image plane. This primarily involves joints with only a limited contribution to the x/y position and is therefore expected to yield a small error.

2. Experiments denoted with "dummy2" use random motions with a standard deviation of σ=80 mm in the x/y-plane. They are therefore expected to suffer from the full scale of the positioning inaccuracy.

Figure 5.31 shows the results for this experiment. As expected, the "dummy1" experiments have little effect on the system error. Using the same scales as in figure 5.30 no noteworthy

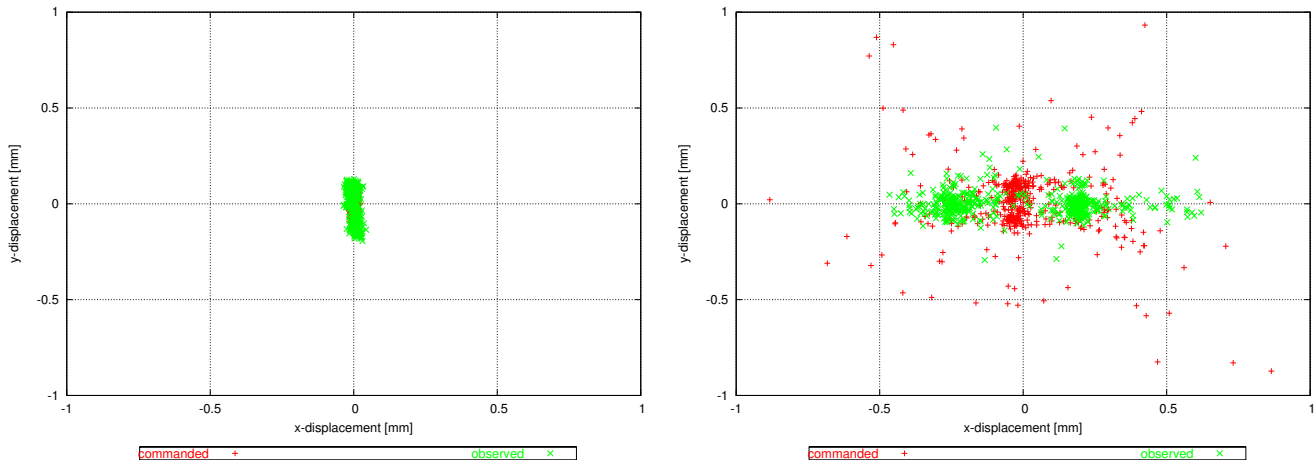
difference to the null motions can be seen, because this motion primarily involves joints that contribute to the z position above the image plane. The "dummy2" experiments do show a significant impact on the system error. Both the arm and the vision error are much larger, because these motions primarily involve joints that contribute to the x/y position in the image plane. Interestingly, the arm error ("commanded") appears to be larger than the vision error ("observed") and the vision error appears to aggregate around two cluster points rather then just one. These effects may occur due to a superposition of limited joint position encoder resolution, gear backlash and the high-level arm control software. None of the effects causes errors beyond what is tolerable, and so they have not been further investigated.

Table 5.2 gives a numerical evaluation of the experiments. For the "dummy1" experiments the arm positioning error is almost the same as with the "null" experiments, which confirms that the motion does not affect the x/y plane much. The vision error is due to the added noise slightly higher, but still its standard deviation is always below 1/10 millimetre. For the "dummy2" experiments the arm and vision errors are much higher than that, with standard deviations of up to 9/10 millimetre. This confirms that the motions do affect the x/y plane. The fact that the vision errors are actually smaller than the arm errors can also be nicely seen.

Experiment	Subtype	σ_x	σ_y	Scaling
20031201-planar/dummy1	commanded	0.008777	0.042649	–
	observed	0.013582	0.089193	0.39
20031201-planar/dummy2	commanded	0.905021	0.746743	–
	observed	0.252359	0.067033	0.39
20031201-tilted/dummy1	commanded	0.018847	0.017677	–
	observed	0.024748	0.097002	0.40
20031201-tilted/dummy2	commanded	0.911211	0.749816	–
	observed	0.235738	0.083085	0.40
20031210-centrifuge/dummy1	commanded	0.029098	0.014753	–
	observed	0.072109	0.038201	0.53
20031210-centrifuge/dummy2	commanded	0.906363	0.749119	–
	observed	0.325007	0.151336	0.53

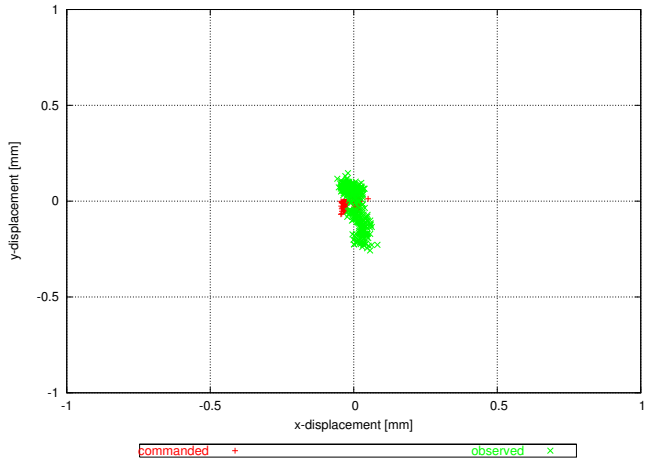
Table 5.2: Standard deviations σ_x and σ_y for the experiments in figure 5.31. All σ values are in millimetres, the scaling factors in millimetres per pixel.

Generally speaking the error values are still below the threshold where they can cause problems for the system and can therefore be neglected. The arm hard- and software itself has a repeat accuracy of about 1 millimetre, so if a sub-millimetre accuracy should be required, special care has to be taken. One possibility demonstrated by these experiment is to split the approach to a target so that the last motion is perpendicular to the plane in which a displacement is expected. Since the usual way to grasp objects in this system is precisely that, a positioning accuracy of only 1 millimetre still safely allows to grasp objects with a precision of less than 1 millimetre.

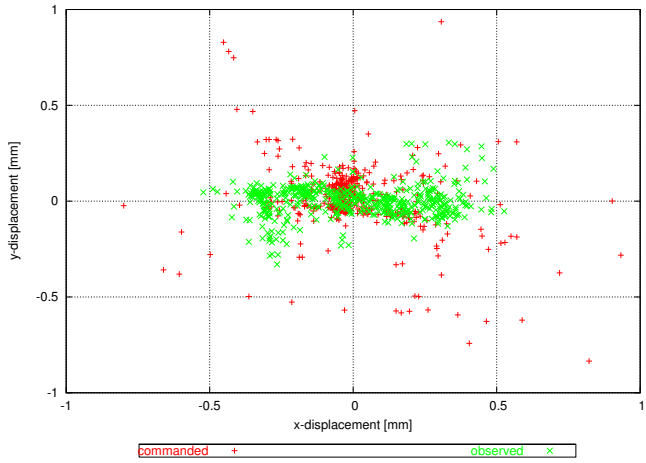


(a) 20031201-planar/dummy

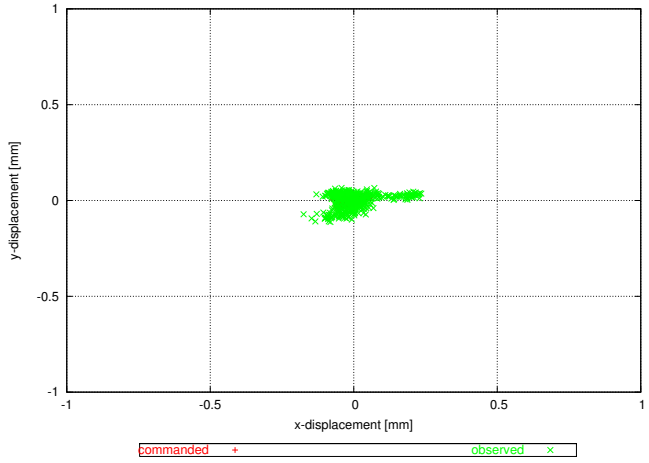
(b) 20031201-planar/dummy2



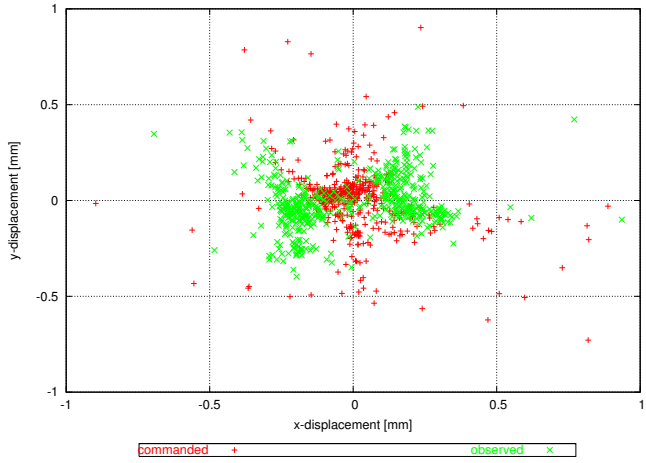
(c) 20031201-tilted/dummy1



(d) 20031201-tilted/dummy2



(e) 20031210-centrifuge/dummy1



(f) 20031210-centrifuge/dummy2

Figure 5.31: Scatter plots demonstrating the effect of dummy robot motions. The rows show the experiment on three different objects: A planar marker, a tilted marker and the centrifuge (which has a tilted front plate). The left column shows a dummy motion perpendicular to the image plane and the right column a dummy motion in the image plane.

5.5.1.3 Influence of Vision Parameters

Another experiment researches the influence of the most important vision parameters on the results. With no standardised illumination the image quality can vary greatly and the images be quite noisy, therefore image noise is an important parameter. Another important parameter is the threshold used in the fill algorithm, which determines the quality with which the regions are detected. Whereas the image noise can only be indirectly adjusted by smoothing the image, the threshold can and has to be directly adjusted. On the other hand, it has to allow a safe operation over a wide range of input image quality, so it must not be too critical. The purpose of this experiment is to establish the influence of these two parameters on the system output.

In the first experiment the influence of image noise on the system output is examined. Images taken during an experiment with "null" motions are artificially superimposed by noise to simulate a degrading image quality. The noise added is zero mean gaussian noise with 0, 3, 6 and 9 pixel values standard deviation for each YUV component individually[16]. By using really different images and not simply adding a different noise to the same image over and over again the situation is similar to the experiment with null motions, and the the resulting error can not be smaller than the basic repeat accuracy.

Figure 5.32 shows the results for this experiment. As can be seen the comparability with the basic repeat accuracy is confirmed. The standard deviation of the error for low noise images is about 1/100 millimetre – the same as with plain null motions. With larger noise the error increases to about 3/100 millimetre. Obviously the noise leads to changes of the segmentation of the regions that affect the COGs, and therefore the output quality degrades.

In the second experiment the influence of the fill threshold on the system output is examined. Applied to the colour similarity images the threshold determines the lower bound down to which a region is filled. A high threshold means that only very few pixel around the initial seed will be found and a large corona around that area will be left unrecognised. Since parts of the corona are likely to be taken as additional regions in a later step this leads to an increased number of regions with no guarantee that the COG of one of them is identical to the COG of the entire region. A low threshold, on the other hand, means that a region will be filled to its very border, with the danger of bleeding out into the background. This means that the COG of that region will shift and therefore yield to problems with the LSF. Both cases should be avoided because the transition from suboptimal classification to complete failure is usually abrupt. As a result of this the threshold is required to work over a wide range without causing this problem.

Figure 5.33 shows the results for an experiment with thresholds of 70, 100 (the nominal values), 130 and 160. As can be seen the vision error is not affected by the threshold at all and stays at the level of the basic repeat accuracy. Only with the highest threshold value of 160 the error seems to increase slightly, but still only within the range of measurement accuracies. The reason for this independence is that a higher threshold leaves a corona around the region, but since for the features used this is a symmetric loss it does not affect the COG and therefore has no influence on the vision system.

Table 5.3 gives the numerical evaluation for these experiments. It can be summarised that the effects of image noise on the vision system are roughly the same as those of dummy motions perpendicular to the image plane. An effect of the fill threshold can even be denied for the

[16] Starting with a standard deviation of 9 the system fails to recognise some images, so this is almost the upper limit. Real images do not have that much pixel noise.

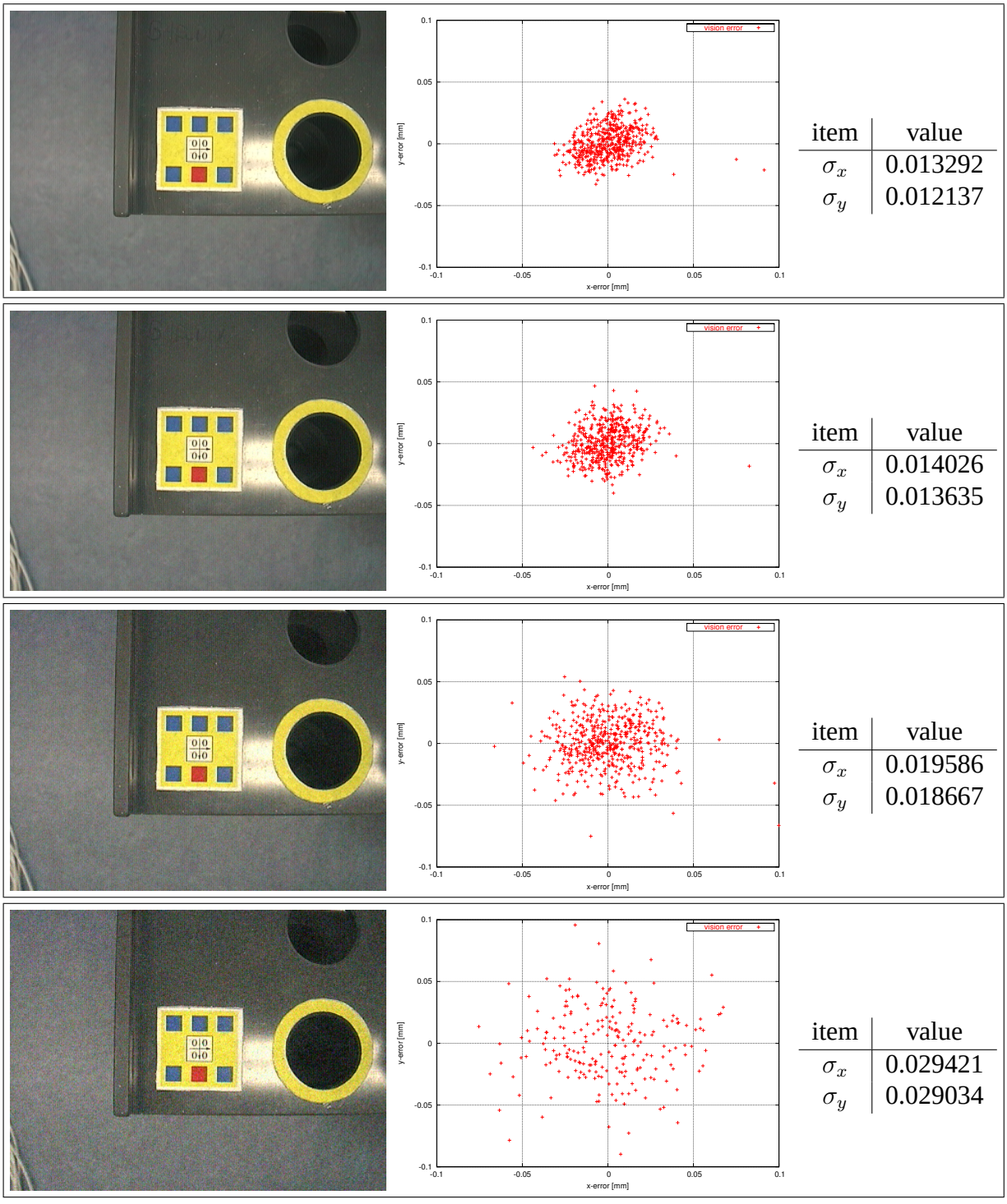


Figure 5.32: Sample images, vision displacement scatter plots and standard deviation (in millimetres) of the distributions for a set of of images with no robot motion and different zero mean gaussian noise levels. The noise levels used from top to bottom are 0, 3, 6 and 9 (where the system starts failing to recognise the object) pixel values standard deviation. As can be seen the noise level does affect the result because it leads to non-symmetric changes of the regions.

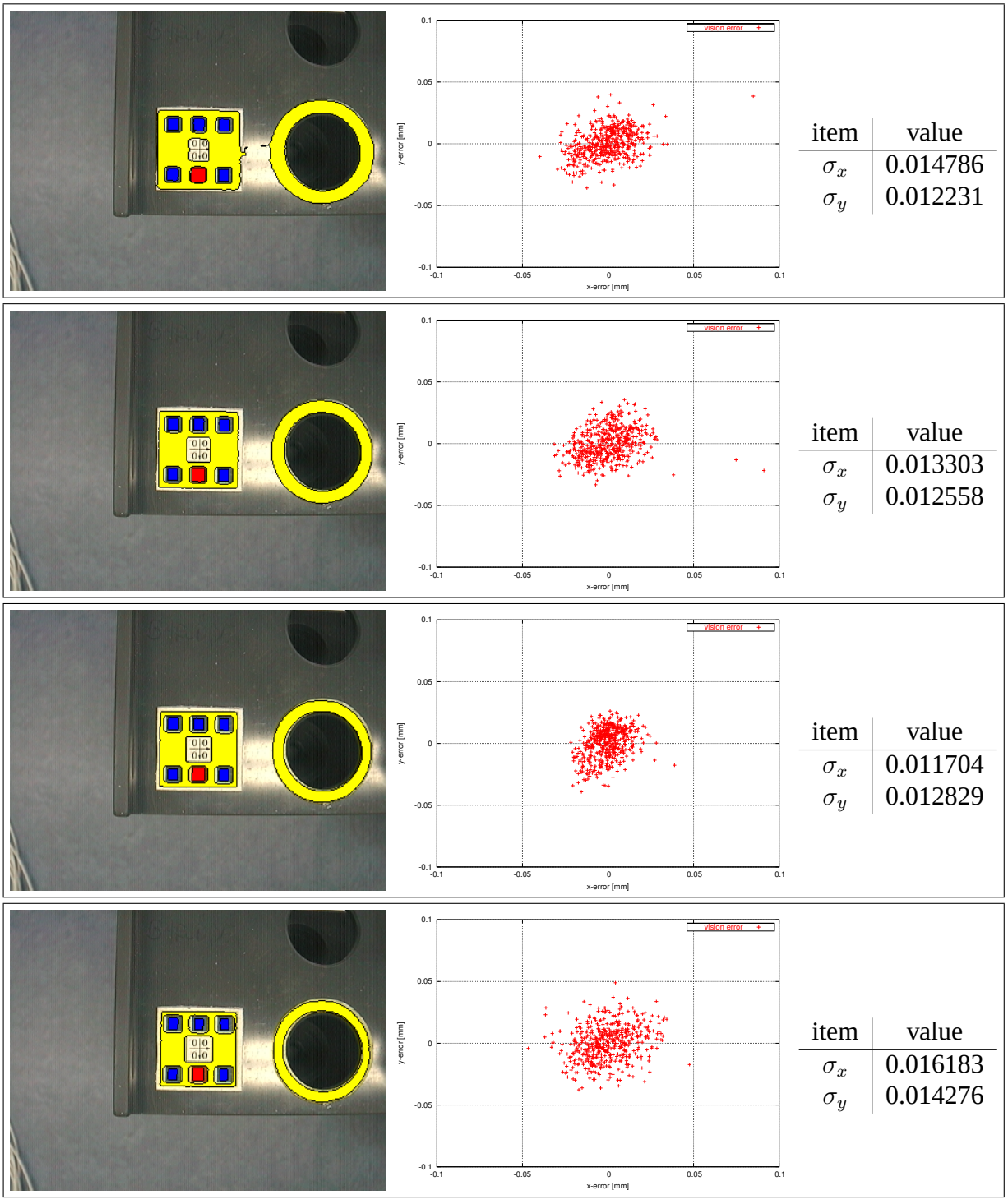


Figure 5.33: Sample images, vision displacement scatter plots and standard deviation (in millimetres) of the distributions for a set of of images with no robot motion and different similarity fill thresholds. The threshold used from top to bottom is 60, 100 (the nominal value), 133 and 166 of a range of 0..255. As can be seen the detected regions become smaller with a larger threshold, but since the loss is mostly symmetric it does not affect the COGs and therefore also not the results.

range of threshold values used. The parameters can therefore be seen as non-critical over a wide range, which allows for an easy selection process.

Experiment		σ_x	σ_y	scaling
20031128-planar/null	commanded	0.010329	0.009896	–
	observed	0.011452	0.012685	0.39
	noise 0	0.013292	0.012137	0.39
	noise 3	0.014026	0.013635	0.39
	noise 6	0.019586	0.018667	0.39
	noise 9	0.029421	0.029034	0.39
	threshold 70	0.014786	0.012231	0.39
	threshold 100	0.013303	0.012558	0.39
	threshold 130	0.011704	0.012829	0.39
	threshold 160	0.016183	0.014276	0.39

Table 5.3: Standard deviations σ_x and σ_y for the experiments in figure 5.32 and 5.33. All σ values are in millimetres, the scaling factors in millimetres per pixel.

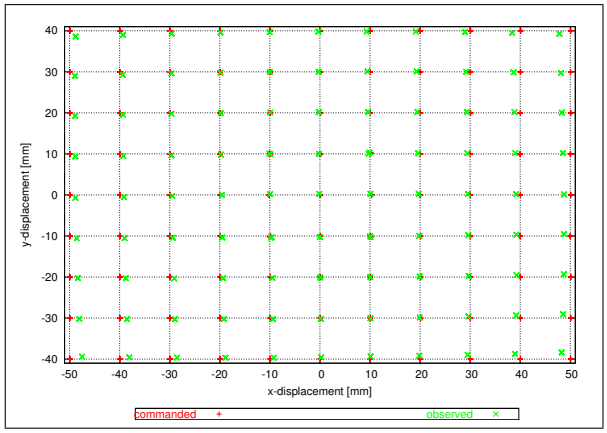
5.5.1.4 Planar Marker Displacement

Having established the basic accuracies and shown that the threshold parameter is not a critical one the errors resulting from the application of the system to real displacements can now be examined. In order to do this, experiments have been done with three different objects in which well-defined displacements according to a grid pattern are applied to the arm. Each displacement is applied several times and the output from the vision system compared against it. The vision error can then be established relative to the displacement. The MSE $e_{C,M}$ used as classification error is analysed too, because with a changing appearance of the pattern it will match only suboptimally.

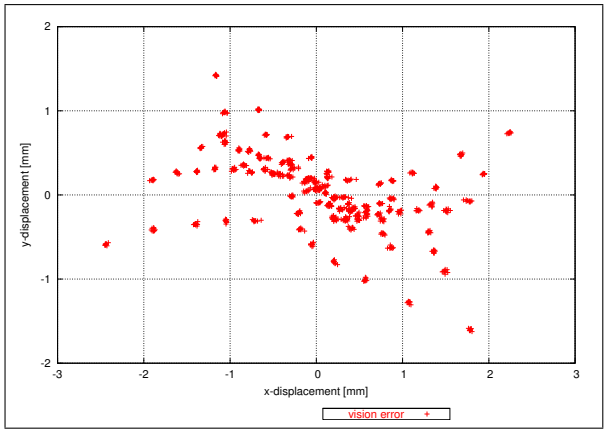
In the first displacement experiment ("20031201-planar") a planar marker on the tubestorage is used and a scaling factor of 0.39 is applied to convert the the vision output into millimetres. Using a planar pattern means that basically only lens errors should affect the output. Since the lens errors are correlated to the radial distance from the optical axis the error is expected to be the higher the larger the deviation is. Overall, the errors resulting from lens errors are expected to be rather small.

Figure 5.34 shows the results from this experiment. The scatter plot shows that for larger displacements the vision computes values which are slightly too small. The vision error, when plotted against the displacement, shows this too. This dependence on the displacement is consistent with the assumption of lens errors. The largest vision error seen is about 2.5 mm at an absolute value of the displacement of about 64 mm, which means that the vision system detects about 96% of the displacement at the worst case. Since the error becomes exponentially smaller with smaller displacements the iterative compensation can be expected to not have any problems with it.

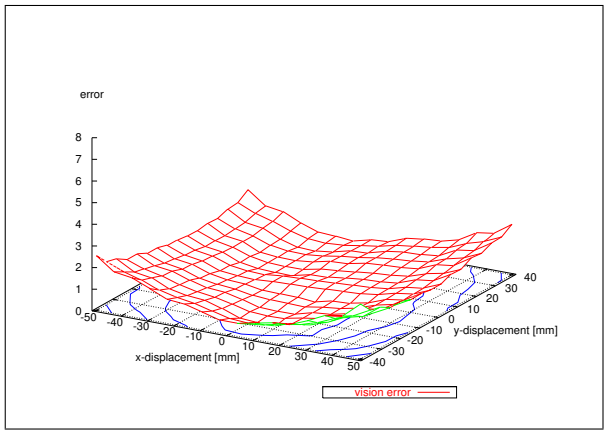
The classification error remains very small over the entire range of displacements. The largest value is about 3 at the largest displacement. It too becomes smaller with smaller displacement, which confirms that the deformation of the pattern correlates to the absolute value of the displacement.



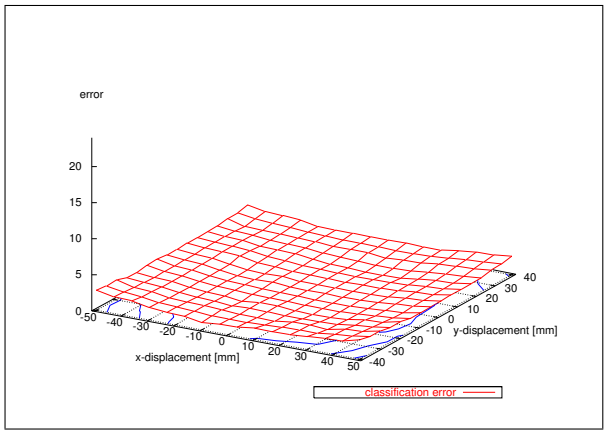
(a) scatter plot



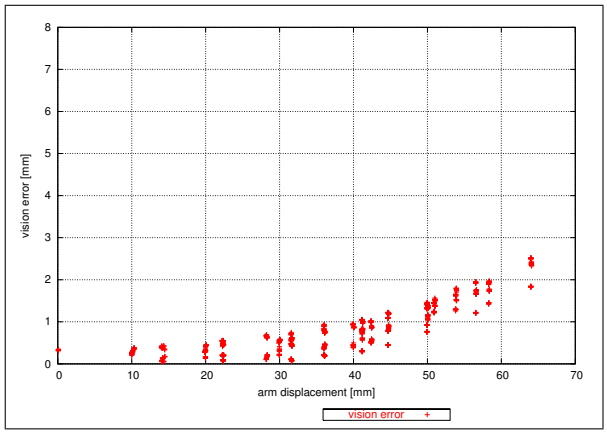
(b) error scatter plot



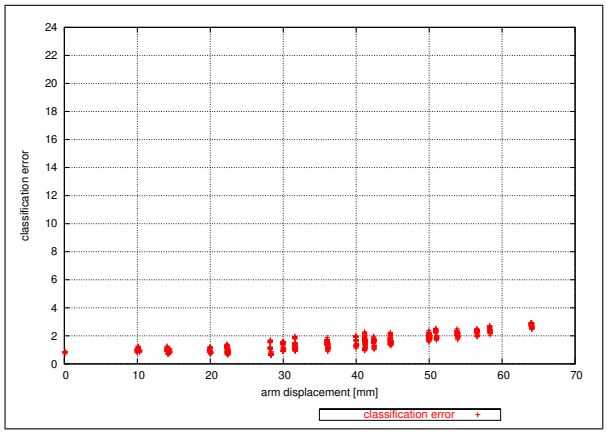
(c) vision error



(d) classification error



(e) vision error relative to absolute value of displacement



(f) classification error relative to absolute value of displacement

Figure 5.34: Results from experiment “20031201-planar”, the translation of a planar marker from the tubestorage. A scaling factor of 0.39 has been used to convert the vision output into millimetres.

5.5.1.5 Tilted Marker Displacement

In the second displacement experiment (“20031201-tilted”) a tilted marker is used. Since the tilted tubestorage used for this requires a slightly larger height for the operation a slightly larger scaling factor of 0.40 is applied to convert the vision output into millimetres. Using a tilted pattern means that in addition to lens errors perspective errors become effective. These, too, are correlated to the radial distance from the optical axis and therefore the shape of the error is expected to be the same. The value of the errors is in turns expected to be much higher.

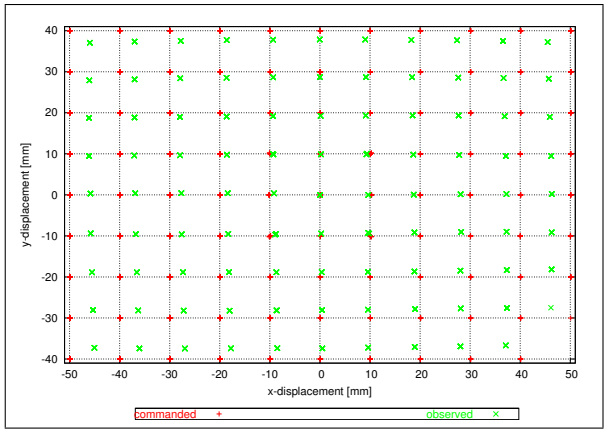
Figure 5.35 shows the results from this experiment. The scatter plot shows the same effect of computing values that are too small as with the planar marker, only that the error is larger this time. This is confirmed by the plot of the vision error against the displacement. The largest vision error seen is about 5.6 mm at an absolute value of the displacement of about 64 mm, which means that this time the vision detects only about 91% of the displacement at the worst case. Since the error still becomes smaller – though apparently more linear than exponential – with smaller displacements the iterative compensation still can be expected to not have any problems with it.

The classification error, too, is larger than with the planar marker and shows an interesting effect: The error does not depend on the absolute value of the displacement, but on the y-value of the displacement only. In addition to this, it does not become smaller with smaller displacements, but converges to a value of about 9.5 only. This can be explained by the fact that in order to do this experiment no explicit classificator for the tilted tubestorage was used, but the classificator for the planar tubestorage was applied to the tilted tubestorage. This classificator of course expects a different appearance of the marker and therefore can not yield a zero error. The interesting point is that this deliberate error does not have a larger impact on the vision output, because it still computes feasible values.

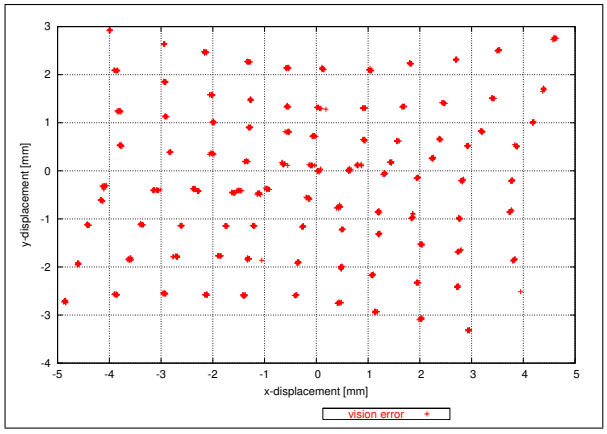
5.5.1.6 Centrifuge Displacement

In the third displacement experiment (“20031210-centrifuge”) the centrifuge is used. Since this again requires a larger height for the operation a larger scaling factor of 0.53 is applied to convert the vision output into millimetres. The difference to the tilted tubestorage is that the pattern on the front plate of the centrifuge is spread over a larger area, which means that it is more affected by perspective errors. The value of the errors is therefore again expected to be higher than in the two previous experiments. Since the tilt angle is also higher the pattern can become invisible if the displacement is large enough so that the camera looks parallel to rather than on the front plate of the centrifuge. For this reason the displacement grid used in this experiment is not symmetric.

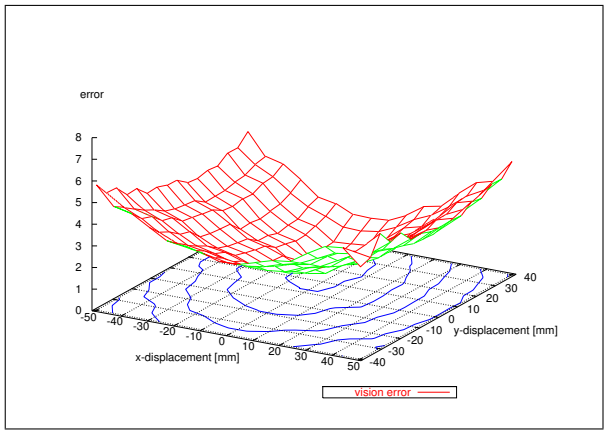
Figure 5.36 shows the results from this experiment. Basically, all plots show the expected behaviour, that means, the vision and classification errors increase with larger displacements. The largest vision error seen is about 7 mm at an absolute value of the displacement of about 85 mm, which means that this time the vision detects almost 92% of the displacement at the worst case. Since the displacement for this worst case is much larger than the displacement for the worst case of the titled tubestorage it can be assumed that the detection rate for the tilted tubestorage would have been larger if a correct classificator had been used. But since again the error correlates with the displacement the iterative compensation can be expected to not have any problems with it.



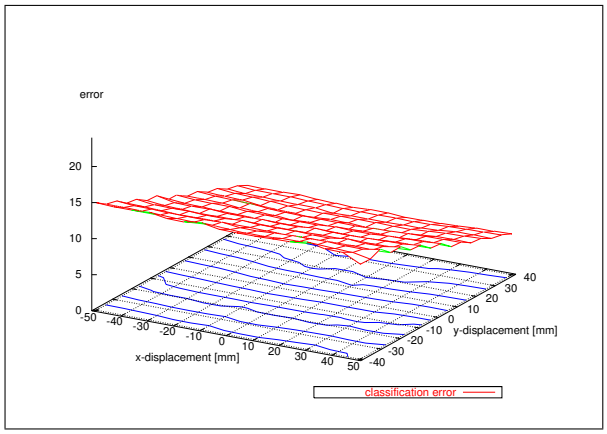
(a) scatter plot



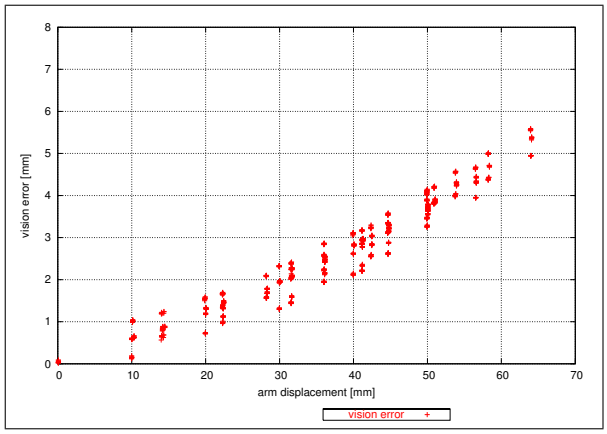
(b) vision error scatter plot



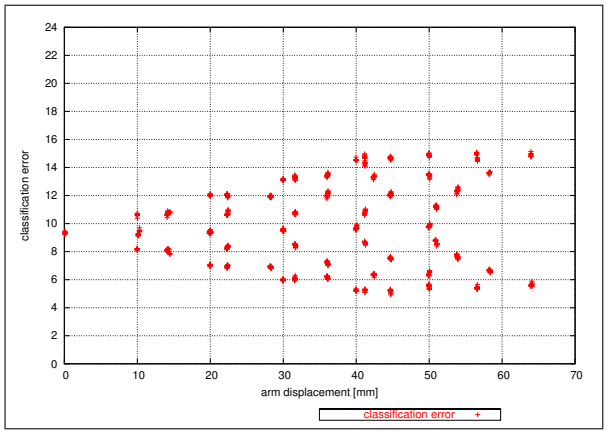
(c) vision error



(d) classification error

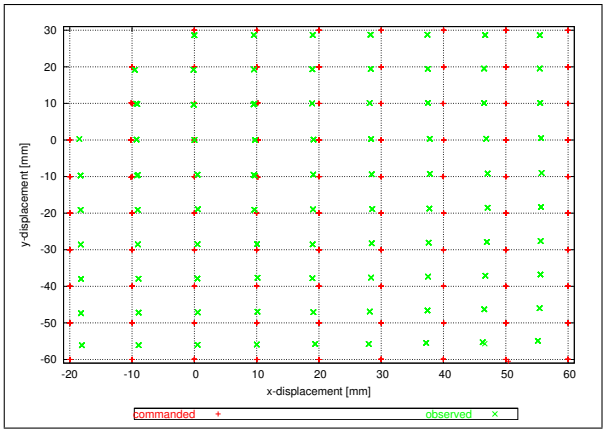


(e) vision error relative to absolute value of displacement

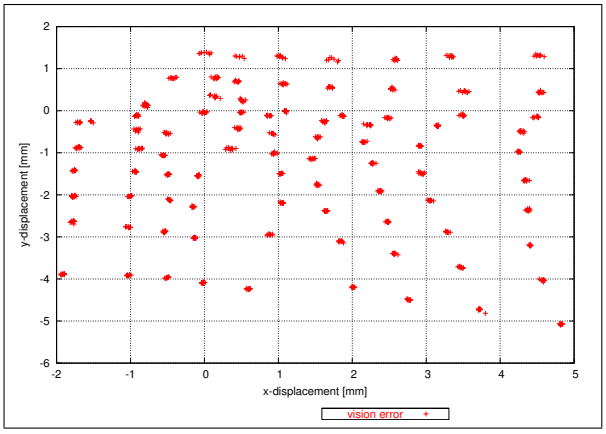


(f) classification error relative to absolute value of displacement

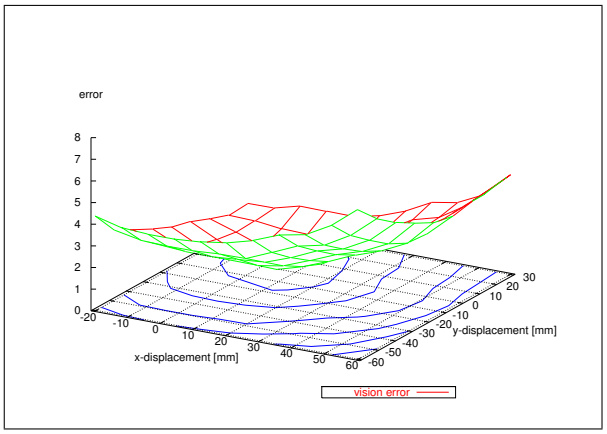
Figure 5.35: Results from experiment “20031201-titled”, the translation of a tilted marker. Since the tilted tubestorage used for this requires a slightly larger height for the operation a slightly larger scaling factor of 0.40 has been used to convert the vision output into millimetres.



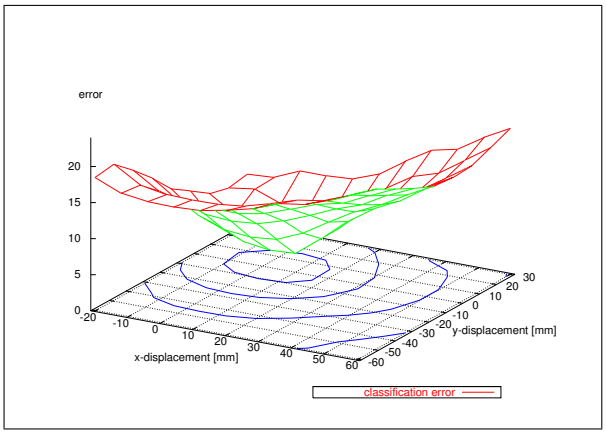
(a) scatter plot



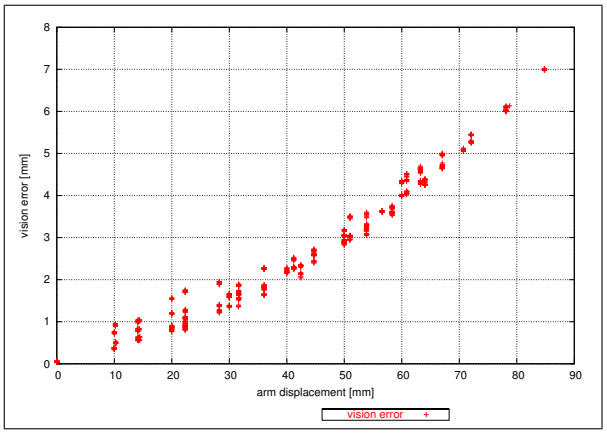
(b) vision error scatter plot



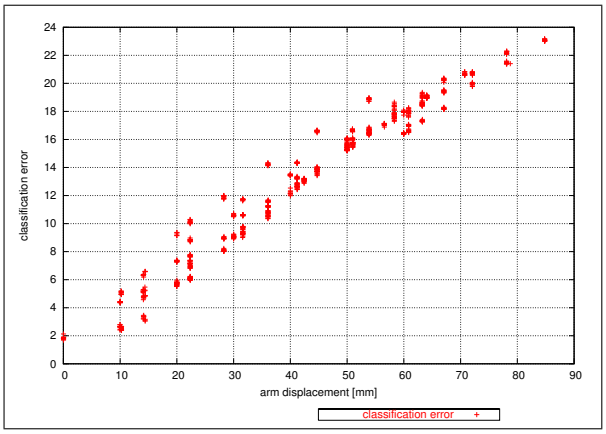
(c) vision error



(d) classification error



(e) vision error relative to absolute value of displacement



(f) classification error relative to absolute value of displacement

Figure 5.36: Results from experiment “20031210-centrifuge”, the translation of a tilted marker from the centrifuge. Since the centrifuge requires an even larger height for the operation an even larger scaling factor of 0.53 has been used to convert the vision output into millimetres.

This time the classification error shows the expected cone shape too, with the cone tip centred at a zero displacement. The largest value is about 23 at a displacement of about 85 mm. Since this value describes the average error for the placement of the individual features of the pattern and since for most patterns used this is actually larger than the nominal distance between two features it becomes doubtful whether the classificator might not start mixing up objects. The apparent correlation of the vision output to the applied displacement denies that, but the problem remains to be further examined in a final experiment.

5.5.1.7 Classification Error Comparison

The above experiments – and in particular the last one – have shown that with increasing displacements the classification error increases too. This raises the question whether it might become large enough to allow a wrong permutation of a wrong object to be mixed with the correct permutation of the correct – but badly weighted – object. To prove the contrary an experiment has been done in which a displacement according to a grid pattern of the size 24 $\times$ 24 millimetres has been applied. The experiment has been repeated on six different objects: The tube storage, the cedex storage, the pipette device, the scanner device, the fridge and the centrifuge (in this order).

Figure 5.37 shows the results from the experiment, where the measurements for each object are sorted according to the increasing absolute value of the displacement and the error has been clipped at 140. As can be clearly seen no wrong positive classification occurs because no classification error of a false object is ever lower than that of the correct one. In most cases the safety margin is several times the error of the correct object, so for most objects no danger of being mixed up exists.

In case of the fridge, however, the margin shrinks to a factor of two compared to the pipette because obviously the patterns are very similar. With an even larger displacement a mix-up might therefore become possible, but this has never been observed in the tests. This problem is not symmetric, because on the other hand side the pipette can be separated from the fridge with a much larger safety margin.

The absolute value of the classification error also needs a discussion. The maximum true positive classification error in the experiment with the displaced centrifuge is 23, compared to the lowest true negative classification error of about 8 in case of the fridge/pipette comparison. Looking only at these two values a correct classification appears impossible and the system can therefore not be called safe. This, however, is not true because the errors have to be seen differently for different objects. An error of 8 is smaller than an error of 23, but in the case of comparing the centrifuge with other objects in fact no true negative classification error lower than about 130 does occur[17]. The error of 8 occurs only when comparing the fridge with other objects, and since the problem is not symmetric it does not affect the system in a way that a false positive classification occurs. Therefore, the values of the errors must only be compared within the correct object class, and using that restriction the system is safe.

The entire question of how safe the classification is under the influence of uncompensated optical effects can be reduced to the question of how similar the patterns are. The markers from figure 5.25 used in this system have not been explicitly designed with an optimised separability in mind, but since they are artificial they could be trivially replaced by improved ones. It can

[17] Also, the classification error of 23 does under real-world conditions not occur because the displacement in the order of magnitude of 85 mm causing it does not occur. See section 3.6 for details.

Figure 5.37: The classification error (MSE) $e_{C,M}$ for six different objects in several different displacements in a 24×24 millimetres grid. The experiments for each object are sorted according to an increasing absolute value of the displacement and the error has been clipped at 140.

happen that the classification fails if one or more of the required regions are missing due to extremely bad environmental conditions, but it has never been observed that it mixes up objects when these conditions are in their normal – quite variable – range.

5.5.2 Summary

The colour vision system established in this work to compensate for object displacements is different compared to other approaches in several aspects:

First of all, the usage of colour information is still an uncommon idea. Classical vision systems used greyscale vision only, because colour cameras were extremely expensive and their image quality not always good (especially for miniaturised cameras) and the computers with the necessary power to perform fast operations on colour images were extremely expensive too. As a result of this, lots of algorithms for greyscale computer vision exist, but only few for colour vision. This situation is currently changing, but the change is not yet fully done. In the meantime a lot of useful information is often wasted by discarding colour information. Instead, since the objects in this work do either have coloured features that can fully represent them or can be labelled with coloured markers for that purpose, an approach that focuses on the colour information and discards everything else has been favoured.

The established colour image segmentation tries to use as much of the colour information while still reducing the dimensionality of the problem as early as possible. It defines and com-

putes a similarity measure of coloured pixels to either a given colour or one selected from the image by its saturation. The measure itself can be expressed as a greyscale image, which means that at a very early stage the complexity is reduced by a factor of three. In these similarity image regions of similar colour are found by using a conventional fill algorithm as a bottom-up strategy. Compared to a top-down strategy like quadtree splitting this approach has the advantage that the outline of regions can be very easily and precisely modelled. An image is then segmented into coloured regions by applying this step iteratively and the regions which are not coloured are discarded. For a typical image of the tubestorage as in figure 5.38, which consists of the marker and one additional region, this process takes about 0.675 seconds on a 2 GHz Intel Pentium 4 processor.

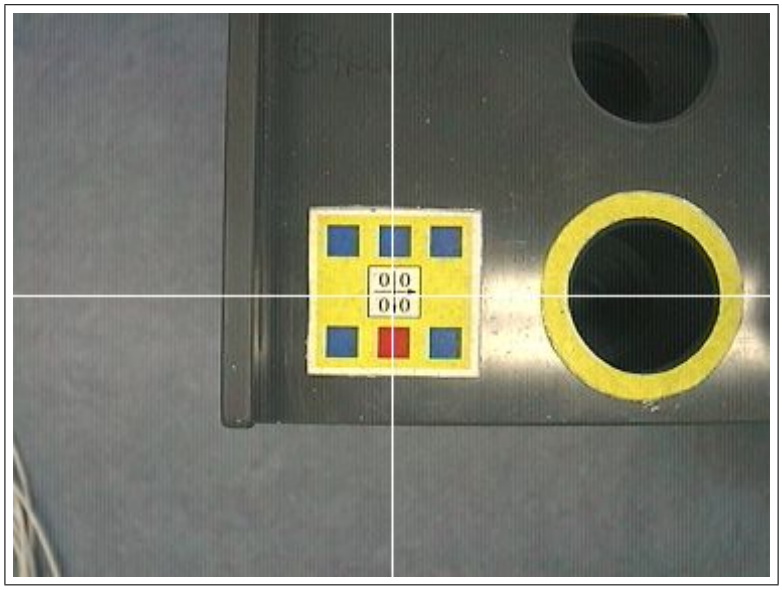


Figure 5.38: A typical image of the tube storage.

These regions are then reduced to their centre of gravity (COG) and colour angle – the characteristic image features. The idea behind this approach is that the segmentation does – depending on the fill threshold and the environmental parameters – not always detect the full object, but leaves a corona around the regions undetected. If only the COGs are used as features and not the entire region this means that the features are largely insensitive against segmentation errors at the border of the region. In case of symmetric segmentation errors as they are likely with symmetric shapes as used in this work they do not even affect them at all. The features are therefore very robust.

From these features a model is generated and stored for each object to be recognised in the desired pose. Object recognition is then performed by evaluating all possible permutations of assignments of image features to model features, because with no initial guess about the position of the object in the image no assumption about the placement of the features can be made. For each of these assignments a 2d-displacement – translation along x and y and rotation about z – is then assumed and the displacement parameters are computed by a least-square fit. The mean-square error of this fit is the classification error and is used to determine the correct assignment and model. For the image in figure 5.38 this process takes an additional 0.019 seconds on a 2 GHz Intel Pentium 4 processor.

The displacement thus computed by the vision system is then used by the arm control software to iteratively adjust the position of the arm until the required accuracy is achieved. This implements a partially uncalibrated approach because neither the lens errors nor the conversion factor between the camera and arm coordinate system are known. It is the explicit design of the vision system to prefer iterations over the requirement of exact calibrations, with the purpose of allowing for easy modifications of as much parts of the setup as possible.

The vision system has been tested in several experiments. According to these experiments the inherent system noise is only about 1 to 2/100 millimetre, caused by image noise and arm vibrations. If the arm is moved this noise increases – depending on the direction of the motion – to about 1 to 9/10 millimetre because of the arm repeat accuracy. Since the arm hardware claims to have a repeat accuracy of only 1/10 millimetre[18] the surplus error must be attributed to the control software. This basic accuracy is affected by additional image noise, but is largely independent from the fill threshold parameter, The latter proves that using only the COGs is a feasible and robust approach.

Experiments with calibrated displacements confirm that the system is an uncalibrated 2-dimensional system because it suffers from lens and perspective errors. Due to these errors the computed translational displacement is computed too low. For planar patterns (a plain 2-dimensional problem) the error introduced by the lens errors is not larger than 4%, and for tilted patterns (a 3-dimensional problem) the combined error introduced by the lens and perspective errors is not larger than 8%. No cases have been observed where the computed deviation was entirely wrong, because if it is due to a wrong assignment of regions this can be detected by limiting the allowed LSF error. This shows that these error are within bounds that still allow a robust compensation in an iteration loop.

The LSF error used for classification allows for a robust detection of obstacles and computation of the displacement in a single step. The markers used to label those objects which do not provide their own feature pattern are not explicitly designed to yield a maximum separability in classification. In a few cases the safety margin between the correct and the next best object class can become small, but so far no false positive classifications have been observed. If the classification fails, it fails because one or more of the regions belonging to the object can not be segmented due to extreme environmental parameters. In case false positive classifications should start to happen with more objects or larger displacements, the markers can easily be redesigned with respect to separability. Therefore the system can be very easily adapted to expanded or otherwise modified tasks.

5.5.3 Future Work & Applications

Despite being a very robust approach, the established system can certainly be further enhanced. Only the biggest of its current drawbacks shall be mentioned here, the computing time needed in the classification. As has been said, with no prior knowledge about the displacement of the object no assumptions about the position of its features in the image can be made. This means that all possible permutations of assignments have to be checked, which yields a $\binom{n}{k}$ complexity. The time of about 0.019 seconds needed for this for a typical image of the tube storage is only a best case, because it has only a single region that does not belong to the object. If an image has a lot of these background regions this time increases over-exponentially.

[18] See section 4.1.1.

One idea to speed up the matching is to try and apply *heuristic* knowledge about the feature placement. Of course, the system is supposed to be able to detect *any* possible displacement, but the hope in many cases is that any *actual* displacement is rather small. With a positioning accuracy of the mobile platform of about 1 centimetre translational and about 1/10th of a degree rotational[19] this seems reasonable. Especially the high rotational accuracy allows to make certain assumptions about the positioning of features, like for example a feature that is left above another feature in the model will also be left above another feature in the image. Using assumptions like these the most likely assignments can be evaluated first, and only if they do not yield a satisfying LSF error entire set of permutations have to be evaluated.

Another idea to speed up the matching is to not compute the LSF for full feature mappings. It has been said that the more features are used the more robust the LSF works, but this is not necessarily true for establishing the mapping too. For example, trying every assignment of image features to only two of the model features yields both a lower number of permutations as well as a faster LSF computation. Given a model with $m \geq 2$ features and a camera image with $c \geq m$ features this reduces the number of permutations to evaluate from $\frac{c!}{(c-m)!}$ to $\frac{c!}{(c-2)!}$. The displacement "guess" computed this way can then be used to map the remaining features, after which the LSF can be repeated with the full model to compute the "real" displacement.

Meanwhile, the idea to use colour imaging is being more widely pursued. The RoboCup[20] championship where teams of mobile robots of all types are programmed to play soccer against each other is but one example. In a typical RoboCup scenario like in figure 5.39 all important parts are coded in deeply saturated colours – in particular the ball. To detect the ball, the goal and the other players a simple colour segmentation like the one developed in this work can be used. With colours as brilliant and distinct as in this example almost no application-specific parameterisation is needed to obtain a good segmentation. As can be seen, the vision algorithms in this work can be applied to a wide variety of situations.

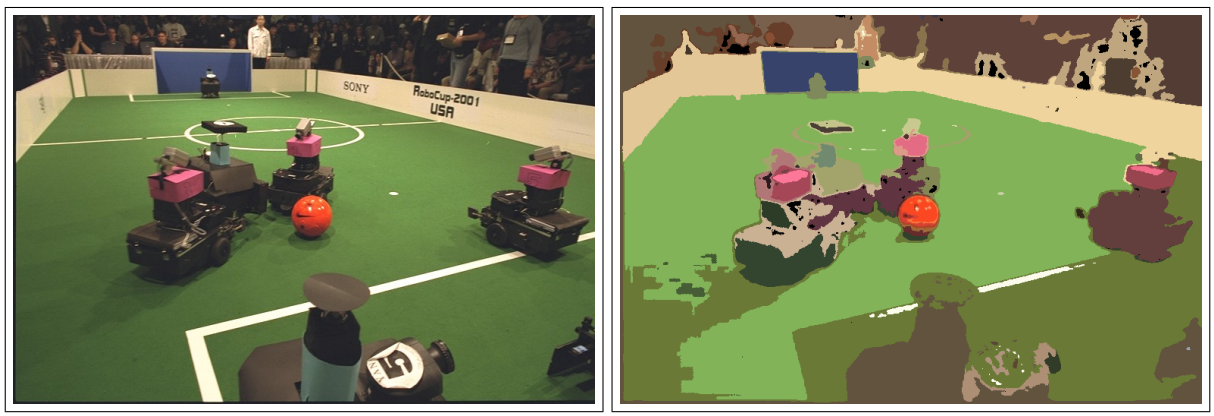


Figure 5.39: A typical RoboCup scenario for the F2000 league. The left image shows the original situation and the right image a colour segmentation with a region growing approach from this work.

[19] See section 3.6.

[20] See `http:/www.robocup.org`.

Chapter 6

Realisation and Integration

"If there's anything more important than my ego around, I want it caught and shot now."

Zaphod Beeblebrox

To implement the functionality presented in the three previous chapters a software architecture given in figure 6.1 has been designed. This architecture consists of several separate programs, each implementing only some aspects of the mobile platform control. The main part is the robot program `robotd`, which is accessed by the *laboratory control program* (LCP). The `robotd` program implements the arm control using RCCL and the software part of the vision system. It uses the `mobiled` program to access the mobile platform, the `imaged` program to acquire images from the camera and the `infobased` program to read parameters and scripts. The `mobiled` can be monitored with a visualisation program called `mobileView`. `robotd`, `mobiled` and `imaged` have to run on the on-board PC of the mobile platform because they need direct access to their hardware, but the other programs can run on any PC in the network. The programs communicate with each other using network support classes.

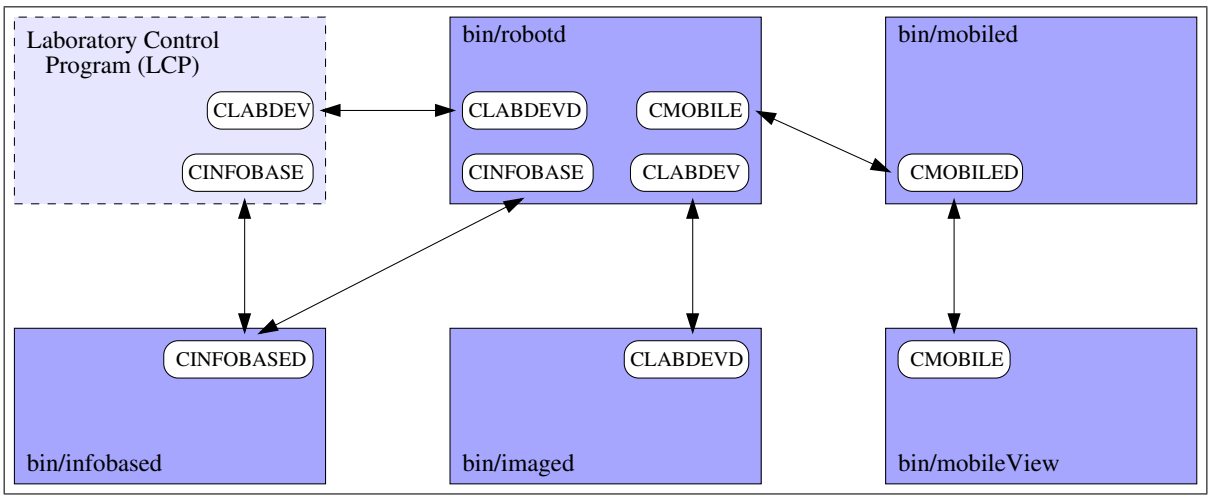


Figure 6.1: Software architecture overview.

In this chapter the major aspects of realisation of the separate modules and their integration into the laboratory context are presented. This includes software details of the implementation

and communication of the separate components as well as aspects which deal with system safety and user interaction and maintenance. The selection is limited to aspects visible or otherwise important to the user. It does, for example, not include the control of the mobile platform itself because this one is completely encapsulated in the main robot application and not explicitly visible to the user. The mobile platform control is therefore only described separately in chapter A in the appendices. The aspects which are described here are:

- The scripting language used for implementing tasks in section 6.1,
- the *deterministic finite automaton* or *state automaton* used to implement safety constraints in section 6.2,
- the database INFOBASE used to store the scripts, state automaton and several other parameters to allow them to be easily modified by the user in section 6.3,
- some general networking issues raised by the interconnectedness of the system and communication software to address them in section 6.4,
- the dedicated network support for laboratory devices to hide the network details from the used as far as feasible in section 6.5 and
- the networked robot as an (actually, *the* most important) example of such a networked laboratory devices in section 6.6,

followed by a short summary in section 6.7.

It is the interface described in section 6.6 that the user sees when accessing the robot system. As such, it is still only usable by a programmer or computer program and not yet an – in terms of specific computer science – unskilled laboratory staff member. Only with its integration into a separate laboratory control system it becomes a fully usable system – a task splitting that was deliberate in the project this work was embedded in. The design and implementation of the laboratory control system is covered in [Poggendorf 2004].

6.1 Robot Command Scripts

Contrary to for example a welding robot used in an automobile factory the situation for the mobile service robot in this work is much more dynamic. In case of a welding robot the situation is static in that it always repeats the exact same operations for weeks and months because its program is usually only changed when a new type of car is to be build. In case of the mobile service robot it is very likely that its program will be changed much more often because of the diversity of the tasks it is expected to perform. Not only the frequency of changes is supposed to be higher, but also the amount of details, because different devices may need entirely different ways to operate them. Therefore the robot software has to offer the user the largest possible range of flexibility – ideally the full flexibility of RCCL.

Even with only a limited knowledge about RCCL (mainly how it handles positions and motions) rather complex sequences of operations can be quite easily programmed, even by non-expert personnel. Using RCCL, however, means programming in C/C++, which is most likely something one does not want the laboratory personnel to have to do. Requiring an – speaking in terms of computer science – unskilled user to learn a full-fledged programming language like

C++ to operate a robot would be overly complicated and only distract the user from its real task. Something easier is therefore needed.

Languages used by commercial robot vendors are typically BASIC-like interpreters (e.g. VAL/V+ from Stäubli) and therefore easier to learn than C++. However, they still require learning a proprietary language, which typically requires expensive training at the robot manufacturer. On the other hand side emulating such a language with all its syntax would be overly complicated.

Other possibilities are existing script languages like `tcl/tk`, `cgi` / `perl`, `PHP`, `python` or shellscripts – the possibilities are limitless, but troublesome. The main problem is that the complete control application can not be realised in such a script language because of the complexity of the task, the separation between what the user may be allowed to edit and what not and especially real-time requirements. The application has therefore been designed to be written in C++, with only some embedded scripts at places where they seem reasonable (the robot motions). However, integrating one of the mentioned script languages into a C++ program is also quite complicated. There exist approaches like `SWIG`[1] or `mktclapp`[2] for `tcl/tk`, but these approaches are not suited to the problems of this work – e.g. they compile static scripts into C/C++, or make C/C++ functions available in `tcl/tk`, which is the wrong way round. And, last but not least, they are also quite complex for a non-expert to learn.

Because of all these disadvantages using an existing language is not feasible. Instead, a simple "parser" for a simple custom script "language" is used that does not share the above problems. It is so downsized in complexity and functionality that it is trivial to learn, yet it is appropriately suited for the task.

6.1.1 Custom Script Language

After having decided to use a custom script language the goal is of course to not make it too complex. The script language therefore does not claim to follow any formal syntax, except that the general structure in EBNF is

```
<COMMAND> [<ARGUMENT>]*
```

The script language includes calling of sub-scripts to reuse existing code, (very) limited conditional branching and limited arithmetics on transforms, some commands to access the encapsulated vision system or the state automaton as well as commands to actually manipulate the robot (motion commands and speed settings).

Variables (i.e. scripts starting with "`s`" and transforms starting with "`t`") are logically bound to a device (including the robot) and loaded on request. They are accessed by names and stored in a stack-oriented way, where calling a (sub-) script enters a new stack context. Searching on this (these) stack(s) is done using a *last in first out* (LIFO) approach, finding the last declared variable of a given name. This means that it is possible to occlude an existing variable by a new one with the same name. When a (sub-) script is terminated the corresponding stack context and all its variables are deleted and "older" variables of the same name become visible again.

[1] `http://www.swig.org`
[2] `http://www.hwaci.com/sw/mktclapp`

A special case are variables that are bound to the robot: These variables are “global” because the stack context of the robot is never deleted. In addition to global configuration they may be used to pass information about special situations between scripts (e.g. `tTargBackup`, which is used to pass the exact position where the robot stopped from `sHoldTubeSampler` to `sTakeTubeSampler`). If changing a script requires exchanging more positional information with other scripts a new transform has to be declared at the robot and the program has to be restarted, however, this is rarely needed and could also be easily changed.

The general execution flow is as follows:

- At program start the parameters and scripts belonging to the robot are loaded and the program enters the main loop, waiting for commands from the network (see section 6.4).
- If a remote client wishes to execute an operation it calls its appropriate local method (e.g. `HoldTubeSampler()`), which sends a command over the network (e.g. `CMD_HOLD_TUBE_SAMPLER`).
- When the command is received the corresponding static top-level script (e.g. `sHoldTubeSampler`) is executed. This script must already exist, so it must have been loaded at program start.
- This script uses the state automaton to check sanity and loads transforms and low-level scripts belonging to the device in question.
- It then calls whatever commands are necessary to implement the operation, either directly or by using subscripts (e.g. `call sApproach` and `call sHoldTube`).
- Upon termination of the top-level script an error code is send back to the remote client, which may then issue other commands.

Because of the reloading of parameters with each command the “real” scripts and transforms finally implementing the sequence can be modified without restarting the program[3].

6.1.2 Script Language Commands

Yorck von Collani has implemented scripts as binary graphs in his OPERA system because this allows easier automatic modifications by learning algorithms[4]. Contrary to that, the scripts in this work are implemented as plain ASCII text because they are intended to be modifiable without a *graphical user interface* (GUI). This does complicate the parsing a little bit, but the advantages due to easy saving/loading, viewing and editing outweigh this. An excerpt of the list of the script commands can be seen in table 6.1.

Contrary to the RCCL++ wrapper around RCCL used by Collani and Ferch these commands do not allow relative movements, but only absolute movements to targets given by position equations or sets of joint angles. If a relative motion is to be performed a transform in a position equation must be changed or a transform added to the equation. This makes the motion primitives look complex, but has the advantage that – using “speaking” names for all transforms – each single motion can be easily verified without requiring the context of previous motions.

[3] In the current implementation this does not hold for the top-level scripts (which is why they are “static”). Changing one of them requires the program to be restarted, but this could rather easily be changed too.

[4] See [Collani 2001].

COMMAND	ARGUMENTS	EXPLANATION
checkstate	[REQUIRES `list`] [CHANGES `list`]	enforce safety and reasonability checks on sequences of high-level functions
call	*name*	invoke sub-script *name*
callif	TRANS | ROT `trsf` *op limit name*	invoke sub-script *name* conditionally
open		open the gripper
close		close the gripper
pushspeed	*scale*	push the current speed on a stack and set the new speed as current speed times *scale*
popspeed		restore the previous speed from the stack
move	`poseq`	move the arm in cartesian space according to a position equation
movej	`poseq`	move the arm in joint space according to a position equation
centerregion	*colour trsf* `poseq`	centre over the already centre-most region of colour *colour* by changing *trsf* in `poseq`
confirmmodel	*device trsf*	confirm model *device* by checking all models and set *trsf* to its displacement
analyzemodel	<device> <trsf>	check only model *device* and set *trsf* to its displacement
centermodel	*device trsf* `poseq`	centre on model *device* by iteratively modifying *trsf* in `poseq`
fmove	[CTRL,`spec`] [ABORT,`spec`] `poseq`	move in cartesian space according to a position equation while obeying force constraints and/or limits
selectslot	*device flags trsf*	set *trsf* to the displacement of a free/full slot from *device* according to *flags*
load	<device>	load parameters and sub-scripts for *device*
settrsf	*trsf* [`coordspec` | *trsf2*]	set *trsf*
newtrsf	*trsf* `coordspec`	create and set a new *trsf* to be visible until the current script is finished
addtrsftrans	<trsf> <transspec>	add *transspec* to the translational part of *trsf*
multtrsf	*trsf trsf2*	multiply *trsf* by *trsf2*
circle	*trsf trsf2 trsf3* `poseq`	move the arm in cartesian space according to a circular motion relative to the current position
mobile	move *device*	move the mobile platform to *device*
mobile	forward *distance*	move the mobile platform forward by *distance* metres (may be negative)
arm	start	start the arm by disabling brakes
arm	stop	stop the arm by enabling brakes
arm	approach `poseq`	sequence of motions to unfold the arm from its park position into an optimal position (in terms of best joint scope) to approach `poseq`
arm	retreat `poseq`	retreat from `poseq` and go into park position by applying the reverse order of commands as in ”approach”

Table 6.1: Excerpt of the most important script commands and their short explanations as currently used for the robot system. Arguments written in CAPITAL are keywords. Variables written in *italics* are atomic symbols, values or names and variables written in `teletype` are composite data structures.

Using the functionality of these script commands the sequence of operations needed to realise the global commands are implemented, and most things can be changed online without having to restart the system. The software has a simulator mode which does not move the arm or the platform and may be used to check if the scripts are logically and syntactically correct and all transforms properly defined.

6.1.3 Example

To give an example of how these scripts work the script to “*place a tube into the sampler and hold it there*” (`sHoldTubeSampler`) will now be presented. The script is bound to the robot and loaded at program start-up. It can be seen (comments stripped and broken into several lines for readability) in table 6.2.

/robot/scripts/sHoldTubeSampler

```
1 checkstate REQUIRES holding=,gripper=nunc,tube=empty
          CHANGES holding=sampler
2 mobile move sampler
3 load sampler
4 call sApproach
5 call sCenterModel sampler
6 settrsf tTargBackup tTarg
7 call sSelectSlot sampler 0 tSlot 250
8 settrsf tSlotBackup tSlot
9 call sHoldTube
```

Table 6.2: The contents of the script `sHoldTubeSampler`.

Line 1 implements the state automaton check described in section 6.2. Line 2 moves the mobile platform to the stored position of the sampler (implicitly updating a global transform `tBase` to the positioning error of the robot base). Line 3 loads the scripts and transforms specific to the sampler (e.g. a transform `tTarg`). Line 4 calls a subscript that performs a basic approach of the arm to the sampler (specified by `tTarg`). Line 5 calls a subscript that performs the vision tasks of classifying the object and compensating its displacement (by modifying `tTarg`). Line 6 stores the value of `tTarg` in the global transform `tTargBackup` because it will be needed by a later script (`sTakeTubeSampler`) and would otherwise be lost. Line 7 selects one of the multiple slots of the sampler rack to place the tube in (by setting `tSlot`). Line 8 stores the values of `tSlot` in the global transform `tSlotBackup`. Line 9 calls a subscript `sHoldTube` that actually performs the operation.

As has been said, line 3 loads scripts and transforms bound to the sampler. These transforms can be seen in table 6.3. Due to the stack mechanism used for variables they will be used for all subsequent commands and sub-scripts until `sHoldTubeSampler` is finished. One example of such a sub-script is the script `sHoldTube`, which is loaded in line 3 and called in line 9. The script `sHoldTube` is used to implement the specific actions needed to hold a tube into the sampler. It can be seen – comments again stripped – in table 6.4.

```
/sampler/trsfs/tTarg
   855.95 139.90 154.61 0.000 90.000 -0.000
/sampler/trsfs/tUpRegrasp
   -10.0 0 0 0 0 0
/sampler/trsfs/tUpTube
   -32.0 0 0 0 0 0
/sampler/trsfs/tRegrasp1
   0 0 -80 0 0 0
/sampler/trsfs/tRegrasp2
   30 0 -80 0 0 0
/sampler/trsfs/tRegrasp3
   30 0 -22 0 0 0
```

Table 6.3: Some transforms bound to the sampling device.

/sampler/scripts/sHoldTube

```
 1 move tBase T6 t7 tGripperTool EQ tTarg tSlot tGraspHeight
 2 settrsf tDown 0 0 0 0 0 0
 3 pushspeed 0.25
 4 fabortmove tDown 6.0 -1.0 0.0 NONE
           tBase T6 t7 tGripperTool EQ tTarg tSlot tDown
 5 popspeed
 6 open
 7 settrsf tDown 100 0 0 0 0 0
 8 pushspeed 0.2
 9 fabortmove tDown 2.0 -1.0 0.0 SOLVE
           tBase T6 t7 tGripperTool EQ tTarg tSlot tDown
10 popspeed
11 move tBase T6 t7 tGripperTool EQ
           tTarg tSlot tDown tUpRegrasp
12 move tBase T6 t7 tGripperTool EQ
           tTarg tSlot tDown tUpRegrasp tRegrasp1
13 move tBase T6 t7 tGripperTool EQ
           tTarg tSlot tDown tUpRegrasp tRegrasp2
14 move tBase T6 t7 tGripperTool EQ
           tTarg tSlot tDown tUpRegrasp tRegrasp3
15 close
```

Table 6.4: The contents of the script `sHoldTube` bound to the sampler.

Line 1 moves the arm to `tBase T6 t7 tGripperTool = tTarg tSlot tGraspHeight` (the `EQ` is motivated by RCCL and just a placeholder for "="), a position equation that that centres the gripper some distance (`tGraspHeight`) over the slot. Line 2 declares and initialises a new transform, `tDown`. Line 3 stores the current speed on the stack and sets the new speed as 0.25 times the current speed. Line 4 moves the arm down with a special force controlled command (note how `tGraspHeight` was exchanged by `tDown` in the position equation from above). The motion is aborted when the force exceeds $6\,N$, whatever the position may be. This "collision" happens when the tube has been fully inserted and hits the bottom of the slot. Line 5 restores the old speed. Line 6 opens the gripper. Line 7 sets `tDown` to a new value (that will actually be inside the rack). Line 8 sets a new speed of 0.2 times the current speed. Line 9 again moves the arm down (because the value of `tDown` has changed), but only until a force of $2\,N$ is exceeded. Contrary to the force controlled move above the value of `tDown` at which this happened is computed for later use (this is what the "SOLVE" says). This "collision" happens when the gripper hits the rack. Line 10 restores the old speed again. Lines 11 - 14 perform motions a little bit up again, a bit backwards, a bit down and a bit forward again, leading to that the gripper now reaches into the rack. Line 15 closes the gripper, leading to that it fixates the tube while the sampler is running.

It can be seen that the operation is basically split into several motions – force controlled ones and normal ones – executed at various speeds. The position equations for these motions are very verbose: Targets are not given directly, but only by means of multiple concatenated relative transforms.

It has to be kept in mind that a position equation does *not* specify a motion, but only a target. The motion is performed implicitly on a straight line from the current position to the target. Therefore, even if a target can be given by a single transform it may only be safely reachable from a certain point with more than one motions. Giving every relative displacement a “speaking” name and not combining the adjacent transforms into something less readable is a good way of making the position equation very intuitive.

It is obvious that - should it become necessary during testing or operation - the sequence can be most trivially edited. Splitting a motion into several motions only means to add lines to the script, and changing targets or relative displacements only means to edit the transforms. No recompilation is needed, and if the changes are not in the top-level script the program even need not be restarted. This way any change (that does not require a completely new command) can be “programmed” very quickly.

6.2 Robot State Automaton

A special command is the `checkstate` command implementing safety constraints. Generally, using the commands from the script language the user can program a great number of sequences, use- and meaningful ones as well as futile and even dangerous ones. A futile sequence would for example be to dispose a tube twice, whereas a dangerous one would be to grab a tube twice.

In the former case there would simply be nothing in the gripper to dispose in the waste bin and the gripper would open and drop nothing, but in the latter case it would already be holding a tube when approaching the storage rack and positioning over another tube. In this case a collision with the storage rack would be unavoidable, and even if it could be detected (which is currently not possible) it would be hard to decide what to do next.

One solution – given a detection was possible – would be to unroll the motions to bring the robot back into a safe position, but that would not recover the complete state information. Appropriate sensors would be needed to tell if the robot has a tube in its gripper after this recovery or not, because otherwise one can not tell if it has slipped during the collision. And even if it still has a tube grasped that tube may have shifted and so subsequent operations are likely to fail. The problem with any collision is that one can not predict its consequences – one can only use sensors to detect them. Since the robot system has only very limited sensoric capabilities this approach would not work and was therefore discarded.

Another and much simpler solution is to deny the command immediately if it does not match the current state of the robot or device, and therefore a *state automaton* has been implemented. In this state automaton the state

$$S \;=\; \{(A_1, V_1), (A_2, V_2), ..., (A_N, V_N)\}\,.$$

is given as a list of *attribute* A_i and *value* V_i pairs. The script for each global command contains a `checkstate` command to check the sanity of the command by *requiring* certain attributes A_i to have certain values V_i and by defining *changes* to them if the command should be legal. Both A_i and V_i are ASCII strings, and therefore S can also be expressed as an ASCII string.

The syntax for the `checkstate` command is defined as

```
checkstate REQUIRES <list> [CHANGES <list>]
```

which means that it expects either two or four arguments:

1. The keyword `REQUIRES`,
2. a list of attribute/value pairs as requirements and optionally
3. the keyword `CHANGES` and
4. a list of attribute/value pairs as changes.

The attributes required and changed do not have to be the same. For example, the `checkstate` statement used in the script for putting a tube into the centrifuge and starting it (`sLoadAndRunCentrifuge`) is given as:

```
checkstate REQUIRES holding=,gripper=nunc,tube=full,centrifuge=open
           CHANGES gripper=,tube=,barcode=,centrifuge=closed
```

The requirements are that the robot must not be standing at any device (`holding=`), must have a full (`tube=full`) NUNC tube in the gripper (`gripper=nunc`) and the centrifuge must be open (`centrifuge=open`). If these requirements are met the operation is allowed and the changes are applied. The changes are that after the operation the centrifuge is closed (`centrifuge=closed`), the robot does not have any tube in the gripper (`gripper=`) and therefore no assumptions about whether that tube is full (`tube=`) or whether it has a barcode (`barcode=`) can be made.

The complete robot state automaton as used in this work is given in table 6.5. The most common attribute `holding` implements the basic safety constraints by allowing only very specific commands in certain situations. For example, in situations where the robot remains standing at a device to hold a tube while that device is operating only the one command removing it from that device is allowed as next command. Other attributes implement more constrains. Using this state automaton some futile sequences are still possible, but the most prominent errors are caught. It is up to the user to impose more restrictions to narrow down the legal actions to a single sequence – if that should be desirable.

The order of the commands in table 6.5 corresponds to one possible meaningful sample management cycle at robot level. The sequence with its state requirements and changes is given as follows:

1. **UnparkCharger:** The robot is unparked from the charging station at which it waits for the beginning of a new sample management cycle. The state requirements before the operation are: The robot must be parking at the charging station (`holding=charger`). The state changes after the operation are: The robot is not standing at any device (`holding=`).
2. **PickTubeStorage:** The robot drives to the storage rack for NUNC tubes without barcodes and grabs one from a filled slot. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must not have anything grasped (`gripper=`). The state changes after the operation are: The robot has grasped an empty (`tube=empty`) NUNC tube (`gripper=nunc`) without a barcode (`barcode=false`).

#	Command	Requirements	Changes
1	UnparkCharger	holding=charger	holding=
2	PickTubeStorage	holding=,gripper=	gripper=nunc,tube=empty,barcode=false
3	HoldTubeSampler	holding=,gripper=nunc,tube=empty	holding=sampler
4	TakeTubeSampler	holding=sampler	holding=,tube=full
5	ShakeTube	holding=,gripper=nunc,tube=full	
6	HoldTubePipette	holding=,gripper=nunc	holding=pipette
7	TakeTubePipette	holding=pipette	holding=,gripper=nunc
8	LoadAndRunCentrifuge	holding=,gripper=nunc,tube=full,centrifuge=open	gripper=,tube=,barcode=,centrifuge=closed
9	PickCedexStorage	holding=,gripper=	gripper=cedex,tube=empty
10	HoldCedexPipette	holding=,gripper=cedex	holding=pipette
11	TakeCedexPipette	holding=pipette,gripper=cedex	holding=,tube=full
12	PlaceCedexCedex	holding=,gripper=cedex,tube=full	gripper=,tube=
13	StopAndUnloadCentrifuge	holding=,gripper=,centrifuge=closed	gripper=nunc,barcode=,tube=full,centrifuge=open
14	HoldTubePipette	s.a.	s.a.
15	TakeTubePipette	s.a.	s.a.
16	PlaceTubeWaste	holding=,gripper=nunc	gripper=,tube=,barcode=
17	OpenFridge	holding=,gripper=,fridge=closed	fridge=open
18	PickTubeStorageBarcode	holding=,gripper=	gripper=nunc,tube=empty,barcode=true
19	HoldTubePipette	s.a.	s.a.
20	TakeTubePipette	s.a.	s.a.
21	HoldTubeScanner	holding=,gripper=nunc,barcode=true	holding=scanner
22	RotateTubeScanner	holding=scanner	
23	TakeTubeScanner	holding=scanner	holding=
24	PlaceTubeAndCloseFridge	holding=,gripper=nunc,fridge=open	gripper=,barcode=,tube=,fridge=closed
25	PickCedexCedex	holding=,gripper=	gripper=cedex,tube=
26	PlaceCedexWaste	holding=,gripper=cedex	gripper=,tube=
27	ParkCharger	holding=	holding=charger

Table 6.5: The complete state automaton with the state requirements and changes for the commands as needed for a meaningful and reasonably safe sample management cycle.

3. **HoldTubeSampler:** The robot drives to the sampling device and holds the NUNC tube into the slot. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped an empty (`tube=empty`) NUNC tube (`gripper=nunc`). The state changes after the operation are: The robot remains standing at the sampling device (`holding=sampler`).

4. **TakeTubeSampler:** The robot takes the NUNC tube it has been holding out of the sampling device. The state requirements before the operation are: The robot must be standing at the sampling device (`holding=sampler`). The state changes after the operation are: The robot is no longer standing at any device (`holding=`) and the grasped NUNC tube is full (`tube=full`).

5. **ShakeTube:** The robot shakes the filled NUNC tube to prevent sedimentation of the cells. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a full (`tube=full`) NUNC tube (`gripper=nunc`). There are no state changes with this command.

6. **HoldTubePipette:** The robot drives to the pipetting station and holds the tube under the pipetting needle. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a (not necessarily full) NUNC tube (`gripper=nunc`). The state changes after the operation are: The robot remains standing at the pipetting station (`holding=pipette`).

7. **TakeTubePipette:** The robot takes the NUNC tube it has been holding out of the pipetting station. The state requirements before the operation are: The robot must be standing at the pipetting station (`holding=pipette`) and must be grasping a NUNC tube there (`gripper=nunc`). The state changes after the operation are: The robot is no longer standing at any device (`holding=`).

8. **LoadAndRunCentrifuge:** The robot drives to the centrifuge, places the NUNC tube into the slot of the cage, closes the lid and starts the centrifuge. The state requirements before the operation are: The robot must not be standing at any device (`holding=`), must have grasped a full (`tube=full`) NUNC tube (`gripper=nunc`) and the centrifuge must be open (`centrifuge=open`). The state changes after the operation are: The centrifuge is closed (`centrifuge=closed`), the robot is no longer grasping anything (`gripper=`) and the fill and barcode flag of the (non-grasped) tube are reset to unknown (`tube=` and `barcode=`).

9. **PickCedexStorage:** The robot drives to the storage for Cedex tubes and grasps one tube from a filled slot. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must not have grasped anything (`gripper=`). The state changes after the operation are: The robot has grasped an empty (`tube=empty`) Cedex tube (`gripper=cedex`).

10. **HoldCedexPipette:** The robot drives to the pipetting station and holds the Cedex tube under the pipetting needle. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a (not necessarily empty) Cedex tube (`gripper=cedex`). The state changes after the operation are: The robot remains standing at the pipetting station. (`holding=pipette`).

11. **TakeCedexPipette:** The robot takes the Cedex tube it has been holding out of the pipetting station. The state requirements before the operation are: The robot must be standing at the pipetting station (`holding=pipette`) and must be grasping a Cedex tube there (`gripper=cedex`). The state changes after the operation are: The robot is no longer standing at any device (`holding=`) and the grasped Cedex tube is now full (`tube=full`).

12. **PlaceCedexCedex:** The robot drives to the Cedex and places the Cedex tube into the slot of the carousel. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a full (`tube=full`) Cedex tube (`gripper=cedex`). The state changes after the operation are: The robot has no longer grasped any tube (`gripper=`) and the fill flag of the (non-grasped) tube is reset to unknown (`tube=`).

13. **StopAndUnloadCentrifuge:** The robot drives to the centrifuge (assuming that it has already finished), opens the lid and grasps the tube from the cage. The state requirements before the operation are: The robot must not be standing at any device (`holding=`), must not have grasped anything (`gripper=`) and the centrifuge must be closed (`centrifuge=closed`). The state changes after the operation are: The centrifuge is now open (`centrifuge=open`) and the robot has grasped a full (`tube=full`) NUNC tube (`gripper=nunc`) without a barcode (`barcode=`).

14. **HoldTubePipette:** see above

15. **TakeTubePipette:** see above

16. **PlaceTubeWaste:** The robot drives to the waste bin and disposes of the NUNC tube. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a (either full or empty) NUNC tube (`gripper=nunc`). The state changes after the operation are: The robot has no longer grasped anything (`gripper=`) and the fill and barcode flag of the (non-grasped) tube are reset to unknown (`tube=` and `barcode=`).

17. **OpenFridge:** The robot drives to the fridge and opens the sliding lid. The state requirements before the operation are: The robot must not be standing at any device, (`holding=`) must not have grasped anything (`gripper=`) and the fridge lid must be closed (`fridge=closed`). The state changes after the operation are: The fridge lid is now open (`fridge=open`).

18. **PickTubeStorageBarcode:** The robot drives to the storage rack for NUNC tubes with barcodes and grasps a tube from a filled slot. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must not have grasped anything (`gripper=`). The state changes after the operation are: The robot has grasped an empty (`tube=empty`) NUNC tube (`gripper=nunc`) with a barcode (`barcode=true`).

19. **HoldTubePipette:** see above

20. **TakeTubePipette:** see above

21. **HoldTubeScanner:** The robot drives to the barcode scanner and holds the tube into its laser beam. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a NUNC tube (`gripper=nunc`) with a barcode (`barcode=true`). The state changes after the operation are: The robot remains standing at the barcode scanner (`holding=scanner`).

22. **RotateTubeScanner:** The robot rotates the NUNC tube it has been holding in the barcode scanner to ensure the laser can read the barcode. The state requirements before the operation are: The robot must be standing at the barcode scanner (`holding=scanner`). There are no state changes with this command.

23. **TakeTubeScanner:** The robot takes the tube it has been holding out of the barcode scanner. The state requirements before the operation are: The robot must be standing at the barcode scanner (`holding=scanner`). The state changes after the operation are: The robot is no longer standing at any device (`holding=`).

24. **PlaceTubeAndCloseFridge:** The robot drives to the fridge, places the NUNC tube in a free slot and closes the lid. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a (either full or empty) NUNC tube (`gripper=nunc`) and the fridge must be open (`fridge=open`). The state changes after the operation are: The robot is no longer grasping anything (`gripper=`), the fill and barcode flag of the (non-grasped) tube are reset to unknown (`barcode=` and `tube=`) and the fridge is closed (`fridge=closed`).

25. **PickCedexCedex:** The robot drives to the Cedex and picks the Cedex tube from the carousel. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must not have grasped anything (`gripper=`). The state changes after the operation are: The robot has grasped a Cedex tube (`gripper=cedex`) with an unknown fill flag (`tube=`).

26. **PlaceCedexWaste:** The robot drives to the waste bin and disposes of the Cedex tube. The state requirements before the operation are: The robot must not be standing at any device (`holding=`) and must have grasped a (either full or empty) Cedex tube (`gripper=cedex`). The state changes after the operation are: The robot has no longer grasped anything (`gripper=`) and the fill flag of the (non-grasped) tube is reset to unknown (`tube=`).

27. **ParkCharger:** The robot drives to the charging station, docks and waits for the next sample management cycle. The state requirements before the operation are: The robot must not be standing at any device (`holding=`). The state changes after the operation are: The robot remains standing at the charging station (`holding=charger`).

It has to be noted that the automaton is a *deterministic finite automaton* (DFA), whereas the system model is *non-deterministic*. The definition of being non-deterministic is that the same action based on the same original state can lead to one of several distinct new states. This can be caused by a random selection of actions or by state information that does not cover all process variables. In the latter case it can happen that an action changes some entity that is not part of the state. The automaton can not see this and therefore can not correctly determine the new state. The usual solution is to use aggregated states and try and separate the possibilities later.

Another way of not being able to determine the correct state is to not have the required sensors. If, for example, the robot is commanded to grasp a tube it can not detect whether the operation has succeeded or not because is does not have a sensor to do this, and any state model of how good something is grasped is therefore useless. Since it has to choose a new state it can as well choose a description of the grip description. Therefore, incomplete observance of the system state automatically makes the system non-deterministic.

Finally, external influence on the system for example by human personnel moving objects can lead to spontaneous state changes. Most of these state changes are observable in terms of being detectable by the robot, but only at the time when the robot hits upon them and not at the time when they occur. External state changes which are not instantly made known to the system make it, by definition, non-deterministic.

All of this is actually the case for the robot system in this work, which does not have enough sensors to monitor all system parameters and can therefore not completely determine its state. The solution in this case is to drop all those components from the state which can not be measured or relied on in the first place. For example, when a tube is grasped the system can not decide whether the operation succeeded or not, it can only assume that it did. The state automaton can therefore not be used to select actions in case operations go wrong, but only to determine legal subsequent operations under the assumption that the previous operation succeeded and its state assumption is correct. Anything beyond this is not within its scope.

6.3 Database

Storing parameters hardcoded in a program is not just bad style but may even be considered a programming error. In case of the system developed in this work and its intended use it would render the entire approach futile because it would require laypersons to edit and recompile the programs after even the slightest changes. Therefore one of the basic concepts of this system is to store *all* parameters in a central data base, including the above mentioned scripts, transforms and state automaton attributes.

6.3.1 Database Considerations

Having decided to use a database for storing parameters, the question of using a commercial available database arises. Due to the inhomogeneous computer hardware with its inhomogeneous operating systems that are required by the different components of the system this database is required to have C/C++ frontends for the Linux, Windows and QNX operating system. This poses a problem because at the time of the system design the commonly known databases like Oracle[5] or IBM DB/2[6] did not support QNX. It is claimed that

only Oracle runs on every popular platform including UNIX, Windows and Linux[7]*,*

but this advertisement (like usual) fails to specify what exactly it means by saying “UNIX”. Other, freely available databases were also discarded because of lacking QNX support. Today there are ways of running e.g. PostgreSQL[8] on QNX, but they require a lot a patchwork to get

[5] `http://www.oracle.com/products/`
[6] `http://www-3.ibm.com/software/data/db2/`
[7] `http://www.ostusa.com/products/oracle_software.asp`
[8] `http:/www.postgresql.org`

it running and are generally unsupported. LDAP[9], the *light-weight directory access protocol*, might have been a considerable alternative, but was not yet commonly known at the time of the system design.

On the other hand side a full-fledged database would be disproportionate for the trivial task of storing simple parameters. Because of this, and because the efforts to port an existing freely licensed database to QNX and to implement a miniature database just suited to the project's needs are roughly comparable a "database" called INFOBASE has been developed.

6.3.2 INFOBASE

The INFOBASE is a small, lean, network-capable client/server database. It is organised like a simple filesystem, storing data in files that are kept in directories. Since it keeps the entire data in memory rather than actually writing it to a real disk it can be seen as a mixture between a ramdisk and a network filesystem. Entry names (directories or files) are treated according to the UNIX convention, i.e. the normal slash "`/`" is used to separate path components. Names may be absolute (starting with a "`/`") or relative to whatever the current working directory is. The virtual directories "`.`" and "`..`" do exist and can be used as in other filesystems, except that the "`..`" directory of the global root directory "`/`" points back to itself. Data is internally treated as binary, although in this project only ASCII data is used. The entire or part of the directory tree can be dumped as ASCII text.

An example can be seen in table 6.6. It contains a directory "robot" which contains the files "address" (no contents specified), "jZero", "models" and a subdirectory "trsfs", which in turns contains a file "t7". The file "/robot/models" shows that the contents may be broken over several lines when printed, but internally it is just a single string. What can not be seen in this print is that the lines 6 and 7 end with a space. This is needed for reading back a dump because the lines are binary concatenated, but the words have to be separable.

```
1  /robot/
2  /robot/address
3  /robot/jZero
4      "0 0 0 0 0 0 0
5  /robot/models
6      "tubestorage tubestoragebarcode sampler sampler2
7      "centrifuge centrifuge2 pipette fridge cedexstorage
8      "cedex cedex2 scanner
9  /robot/trsfs/
10 /robot/trsfs/t7
11     "0 0 80 0 0 0
```

Table 6.6: Example for INFOBASE data storage.

Just like for printing the directory tree (`ibls`), tools exist to remove (`ibrm`), copy (`ibcp`) or move or rename (`ibmv`) entries and to save and load (parts of) the tree (`ibsave` / `ibload`). These tools can be used as a way to modify the contents of the INFOBASE, however, most higher level programs use the provided C++ class `CINFOBASE` implementing the client access.

[9] `http:/www.openldap.org`

In a networked environment the client and server of a database need not be running on the same machine. In case of the INFOBASE the server `infobased` is found by the clients using broadcasts, which means that it can run on any machine that is in the same sub-net as the others[10]. Broadcasts in an IP network are UDP datagrams and do not support features like flow control and fragmenting. Because of this the broadcasts are only used to establish the IP address of the server. After this initialisation step a TCP connection is established between the client and the now known server. This method of locating the INFOBASE is used by the `CLABDEV` class described in section 6.5 to provide transparent network support for other components of the system.

The clients access the INFOBASE server `infobased` by means of a C++ client class `CINFOBASE` that is documented in section D.1 in the appendices.

6.4 Networking

The INFOBASE is only one component of the system that has to be accessible by other components. Overall, the system consists of many components implemented on or accessed by many computers using many different communication media and protocols. The resulting network is very complex, as can be seen in figure 6.2. For the purpose of this work the individual devices can be grouped into three classes:

1. "Passive" devices that do not need and do not have any computer interface at all, but are only passively operated. Examples include the storage racks and the fridge.

2. "Active" devices that do have an interface to offer at least some low-level functionality. Because of their limited capabilities these devices actually *require* an external controlling computer to perform complex and useful tasks. Examples include the first sampler where stepper motors have to be controlled over a Centronics parallel line in real-time, and the second sampler and the pipetting station which have to be programmed by RS422/RS232 serial lines.

3. "Intelligent" devices that can be given complex commands and have enough intelligence to process them on their own. These devices have an ethernet interface and can be accessed by TCP/IP. Examples include the third sampler and the robot.

This section describes the communication of the type 3, the TCP/IP networked devices only. The details of the other communication media and protocols is discussed in [Poggendorf 2004].

6.4.1 TCP/IP Networking

The system does use distributed hardware, but the distribution is somewhat limited. The type 2 devices have to be tightly connected to a computer acting as a master for them, which automatically leads to a centralised system because due to hardware and simplicity reasons exactly and only one computer is used for this. Due to these technical constraints the communication is done in a master/slave manner. Complex communication networks as contract nets used by C.

[10] Contrary to multicasts, broadcasts are for obvious reasons not forwarded by routers and switches, therefore only machines in the same sub-net see the broadcast. In case the INFOBASE should be running on a machine that can not be reached by broadcast the environment variable `INFOBASE` may be set to an explicit host name. If this variable is present the broadcast mechanism is overridden.

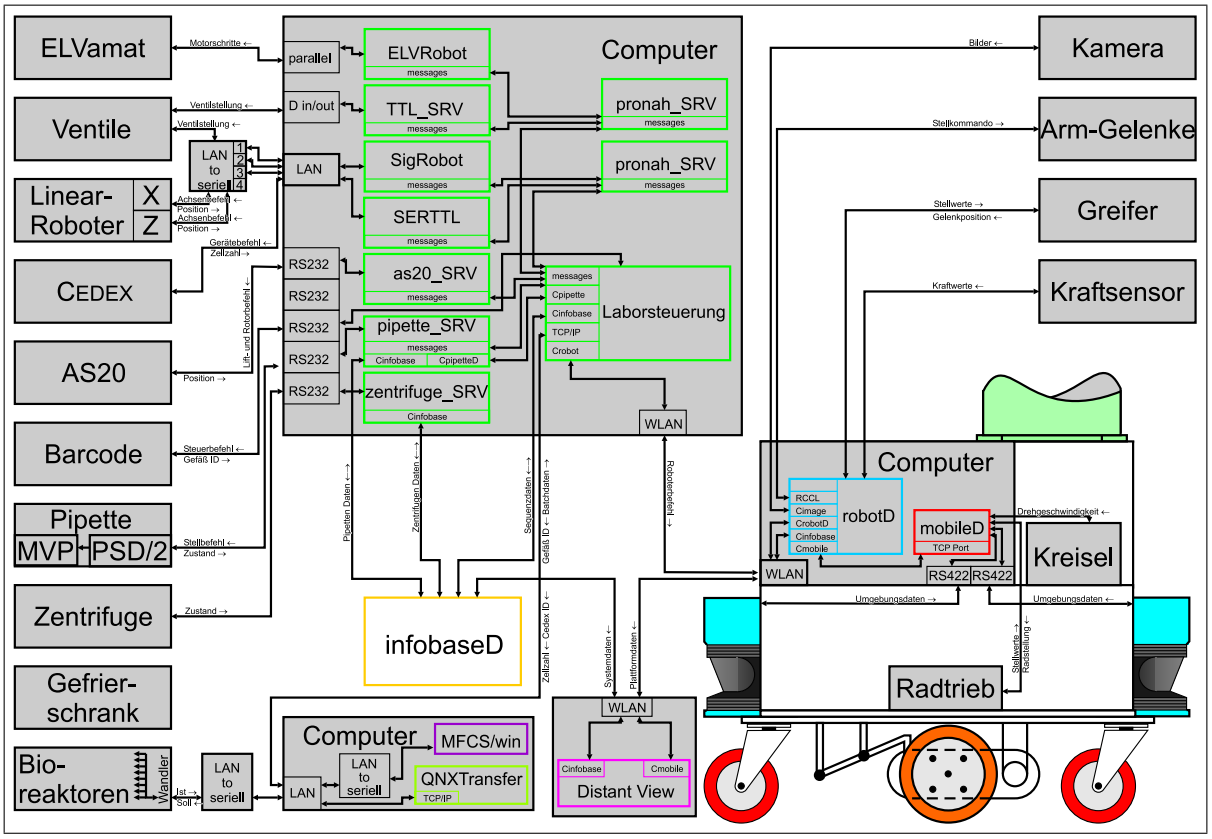


Figure 6.2: Overview of the interconnections and the data flow between the different components of the entire robot system. Image by I. Poggendorf.

Scheering[11] and T. Graf[12] or other (multi) agent systems as used by M. Ferch[13] are not required by the task definition.

This motivates organising the TCP/IP networked type 3 devices according to the same idea – a dedicated master/slave architecture. Logically, the resulting network as in figure 6.3 therefore has a *star topology* using a TCP/IP medium as backbone. Technically, the main medium is wired 100 Mbps IEEE 802.3u ethernet, but the mobile robot can of course not use wires. It is therefore attached to the network by 11 Mbps IEEE 802.11b *wireless lan* (WLAN), using a dedicated PC as router into the wired network. WLANs, however, do raise quite a few issues like their operational range or safety questions.

The operational range of a IEEE802.11b WLAN is typically 100 m in an open-air environment and may be less than 10 m inside buildings with concrete walls and/or other obstacles. This is currently not a problem because the task does not specify leaving the room in which all the devices are cumulated. If leaving the room should become desirable a more sophisticated WLAN with several *access points* could be used to extend the operational range.

Another issue is the safety of a WLAN – since everyone within the operational range can intercept the radiowaves everyone can read the raw data being transmitted. The *wired equivalent privacy* (WEP) used in IEEE802.11b networks to encrypt the raw data can not be considered

[11] See [Scheering 2000].
[12] See [Graf 2000].
[13] See [Ferch 2002].

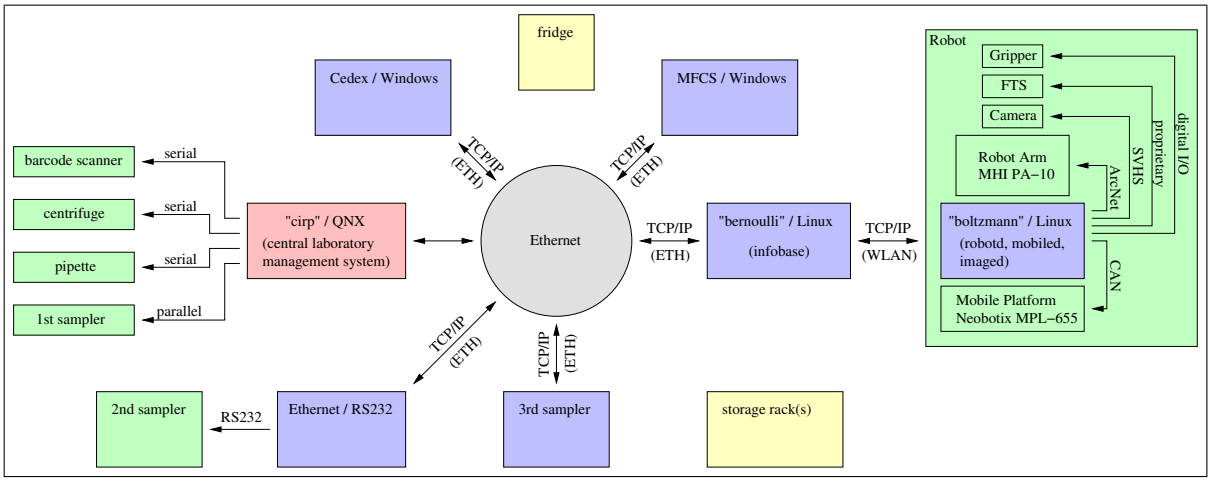


Figure 6.3: The network oriented system setup. Items marked in blue are computers or intelligent devices that can be accessed via ethernet. The box marked in red is the central controlling computer "cirp" running the laboratory management system. Items marked in green are devices than can only be accessed indirectly and items in yellow do not have a computer interface at all.

safe because it has known systematic weaknesses – working exploits do exist. Since the robot is not yet used continuously this is a suboptimal situation, but not a real safety risk. For a more permanent use the safety should of course be improved by implementing *access control lists* (ACLs) on the bridge computer, allowing only known clients to use the network, and/or by using *virtual private networks* (VPNs) realizing a safe high-level encryption.

6.4.2 CNETOBJ

In order to simplify the interaction with the TCP/IP network a framework called `CNETOBJ` has been developed that hides network details and allows for efficient and simple transfer of C++ objects. It deals with properly encoding object data and creating proper object instances on the receiving side. As such it is similar to frameworks like CORBA[14], but it can not be really compared with it because it aims at only a fraction of CORBA's goals. The main purpose of the `CNETOBJ` framework is to allow a simple, lean and efficient one-shot master/slave object transfer of C++ objects between hosts in a common subnet, and not the full feature set of CORBA, which is a more abstract definition supporting multiple programming languages, network topologies and communication types. `CNETOBJ` is therefore very small and can be used on system with only very limited resources, whereas CORBA involves a lot of overhead that requires more powerful machines.

The `CNETOBJ` communication and transfer mechanism consists of two parts, the `CCHUNK` class implementing a buffer and the `CNETOBJ` class itself. Both classes are documented in section D.2 in the appendices.

6.5 Network Support for Laboratory Devices

Based on the low-level `CNETOBJ` mechanism is the mid-level network support for laboratory devices implemented in the `CLABDEV` framework. In addition to the data encoding and transmission of `CNETOBJ`, this framework presents an encapsulation of services that further simpli-

[14] See `http://www.corba.org` and the Object Management Group at `http://www.omg.org`.

fies the network use. It implements a client/server based one-shot service where exactly one command (with parameters) is send to a server and exactly one answer (with parameters) is send back to the client.

Instead of accessing a remote server by its IP address and port number directly they access it by its unique *name*. To allow clients to resolve these names the server stores its IP address and port number in the entry

`/`*name*`/address`

in the INFOBASE, leading to a communication as in figure 6.4. Since the INFOBASE can in turns automatically be found by the clients using broadcasts all services using the CLABDEV framework can therefore be automatically found regardless of which computer they are implemented on. By resolving the name *each time* a service is called and only establishing a connection for the time of the one-shot service the design even allows to transparently restart servers while the main control program is running.

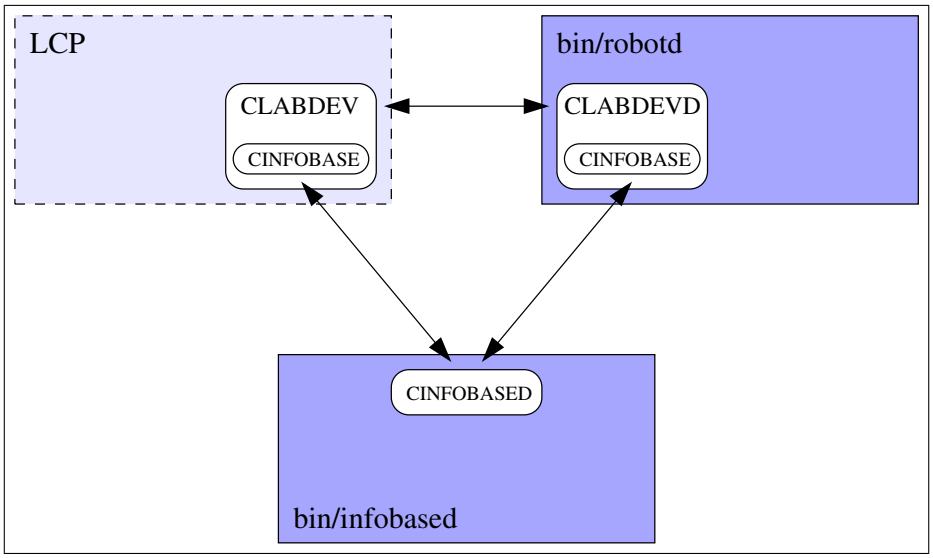


Figure 6.4: The CLABDEV network support framework.

6.5.1 CLABMSG / CLABDEV / CLABDEVD

The CLABDEV framework is implemented in three C++ classes: The message class CLABMSG, the client class CLABDEV and the server (daemon) class CLABDEVD. The message class CLABMSG defines parameters as in table 6.7, which are more than enough for the mobile robot system because only the names of scripts to be executed are transmitted, plus at most two parameters in very few cases. Other parts of the laboratory automation system also using the CLABDEV framework use more parameters, but the current restriction has proven to not be a limitation. The arrays are of fixed size for simplicity reasons only – the CNETOBJ framework could also support transmission of variable sized arrays.

The CLABMSG, CLABDEV and CLABDEVD classes are documented in detail in section D.3 in the appendices.

```
#define MAXSTRINGDATA 4

class CLABMSG : public CNETOBJ
{
public:
  int _cmd;
  int _intData[8];
  float _floatData[8];
  char *_stringData[MAXSTRINGDATA];
  int _auxDataSize;
  void *_auxData;
};
```

Table 6.7: The message class `CLABMSG`.

6.5.2 Laboratory Devices

The C++ classes `CLABMSG`, `CLABDEV` and `CLABDEVD` implement only the abstract mid-level framework of the network communication, but they do not implement any real service. Support for a laboratory device using these classes has to be implemented in derived classes that implement or overwrite the appropriate methods. The following devices currently use this mechanism of communication:

- The MFCS,
- the pipette (although in this case client and server are running on the same host),
- the Cedex,
- the camera server `imaged` and
- the robot server `robotd` described in the next section.

Communication over the `CLABDEV` framework is currently subject to some safety issues. Most notably the only protection used on the WLAN is the weak WEP encryption[15]. At the time of writing cards with the newer and better *wi-fi protected access* (WPA) encryption were not yet supported under Linux. The only protection is the `Allow()` method of `CLABDEVD`, which can limit access to certain IP addresses. While faking IP addresses is not impossible, it is not trivial, especially if the network is properly administered.

As a result, the current implementation should only be used in a restricted environment. If more protection is needed the software can either be changed to use an explicit SSL encryption or a VPN can be established over the WLAN. The latter solution has the advantage that is does not require the software to be changed.

[15]WEP uses the stream encryption method RC4 and computed – but not random – initialisation vectors, which makes it decipherable if only enough ciphered data can be monitored (WEP64 (40 bit key) app. 15 min, WEP128 (104 bit key) app. 40 min)

6.6 The Networked Robot

One – the most interesting – example of a high-level service based on the mid-level CLABDEV framework is the robot server in figure 6.5. The robot server implements the arm control using RCCL and the vision system. It in turns uses the CLABDEV framework to access the image server imaged to acquire images from the camera. Access to the mobile platform control program mobiled is established by a different communication framework because the mobile platform control is designed to be accessible by other programming languages than C++ (see chapter A in the appendices).

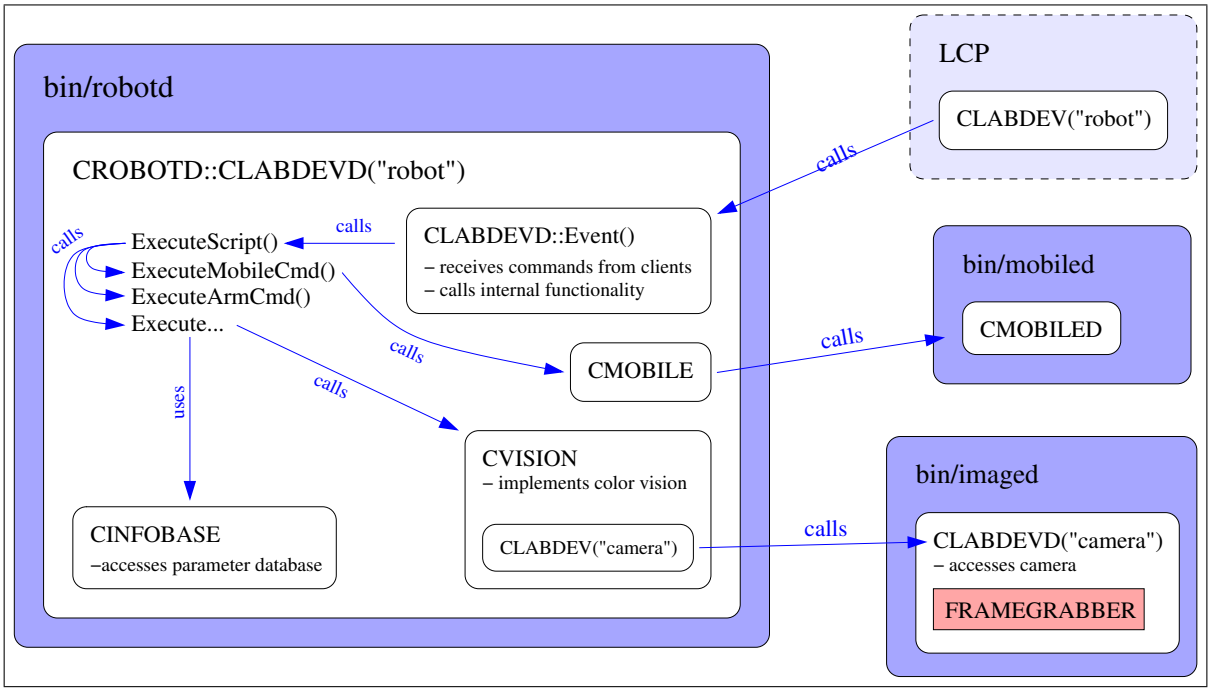


Figure 6.5: The robot server robotd.

6.6.1 CROBOT / CROBOTD

The robot support is implemented in two C++ classes, CROBOT for the client side and CROBOTD for the server (daemon) side. These classes are derived from their CLABDEV counterparts and implement the interface to the robot's functionality. Their full documentation is given in section D.4 in the appendices.

The client class CROBOT is derived from CLABDEV. It implements the client interface to the robot's functionality and is documented in subsection D.4.1 in the appendices. It is all that can be externally seen from the robot – all the implementation details of the commands are hidden in the scripts and parameters in the INFOBASE. An excerpt of the methods implemented by CROBOT can be seen in table 6.8. The methods have quite "speaking" names, and so only the first one shall be described here to show the abstract level at which robot commands operate[16]:

[16] See the definition of the state automaton on page 219, or tables 7.1 and 7.2 in section 7.1 for explanations of the other methods.

```
class CROBOT : public CLABDEV
{
public:

  CROBOT (void) :  CLABDEV ("robot");
  virtual ~CROBOT (void);

  int PickTubeStorage (void);
  int PlaceTubeStorage (void);
  int PickTubeStorageBarcode (void);
  int PlaceTubeStorageBarcode (void);
  int PlaceTubeWaste (void);

  int HoldTubeSampler (void);
  int TakeTubeSampler (void);
  int HoldTubeSampler2 (void);
  int TakeTubeSampler2 (void);

  int HoldTubePipette (int numShake,
                       double depth);
  int TakeTubePipette (void);

  int OpenCentrifuge (void);
  int PlaceTubeCentrifuge (void);
  int PickTubeCentrifuge (void);
  int RunCentrifuge (void);

  int CloseCentrifuge (void);
  int CloseCentrifugeReally (void);
  int LoadAndRunCentrifuge (void);
  int StopAndUnloadCentrifuge (void);

  int OpenFridge (void);
  int PlaceTubeFridge (void);
  int CloseFridge (void);
  int PlaceTubeAndCloseFridge (void);

  int HoldTubeScanner (void);
  int RotateTubeScanner (void);
  int TakeTubeScanner (void);

  int PickCedexStorage (void);
  int HoldCedexPipette (double depth);
  int TakeCedexPipette (void);
  int PlaceCedexCedex (void);
  int PickCedexCedex (void);
  int PlaceCedexWaste (void);

  int ParkCharger (void);
  int UnparkCharger (void);
};
```

Table 6.8: Excerpt from the client class `CROBOT` as declared in `robot/robot.h`. Apart from deriving itself from `CLABDEV` and calling that constructor with its name ("robot") it basically declares some methods implementing the client part of the robot's functionality.

PickTubeStorage() drives the mobile platform from wherever it is standing to the tube storage with NUNC tubes without barcode, moves the arm over the expected position of the marker on the storage, finds the marker using the vision system, compensates the displacement, selects a slot to take a tube from, moves the arm to that slot, uses to vision system to centre on the tube, grasps the tube and retreats the arm into a park position.

Amongst the methods only two take any arguments: `HoldTubePipette()` and `HoldCedexPipette()`. This is because – seen from the robot – the *depth* how far a tube should be held under the pipette and the number *numShake* how often a tube should be shaken for resuspending the cells are the only parameters variable at run-time. For all other cases the parameters are static and stored in the INFOBASE.

The depth has to remain variable because it depends on the fill level of the broth in the tube: For the centrifugated broth the needle must only penetrate short below the surface of the liquid to be sure not to suck any cells from the bottom of the tube. The robot itself does not keep state information (though the state automaton partially does) about whether a tube is centrifuged or not and therefore can not decide this.

The server class `CROBOTD` is derived from `CLABDEVD` and implements the robot functionality. In addition to the network communication it also contains the complete robot arm control[17],

[17]The mobile platform is accessed using the C++ class `CMOBILE`. This class implements a remote control that does not follow the `CLABDEV` scheme. The reason for this is that the `CMOBILE` class was intended to be used in a much wider range of projects which have got nothing to do with the laboratory setup in this project.

making it quite complex. Since the internals of CROBOTD are not visible to the user they are not given here, but are documented in subsection D.4.2 in the appendices.

6.6.2 robotd

The CROBOTD class is implemented in the robotd program – the robot server. The robotd program accepts a number of command line arguments listed in table 6.9. The meaning of these arguments is as follows:

```
tesche@fermi/pts/2> ./robotd -h
usage:  robotd [<option>]
     where <option> is any combination of
     -d :  enable debugging
     -h :  print this help
     -m <hostname> :  connect to mobile <hostname> instead of default
     --no-arm :  disable arm
     --fake-arm :  fake arm movements
     --no-mobile :  disable mobile
     --no-force :  disable force control
     --no-vision :  disable vision
```

Table 6.9: Command line arguments of robotd.

-d enables lots of debugging output. This option is not recommended for normal usage because it can disturb the real-time capabilities of the system due to calls to the write() system call by real-time threads.

-h prints the arguments as listed in table 6.9.

-m <**hostname**> connects to an explicit mobile platform instead of the default localhost. This option is only useful for debugging or experimental setups.

--no-arm completely disables the arm. Includes --no-force and --no-vision. No motions are performed, no checks are done and everything is silently assumed to succeed.

--fake-arm fakes using the arm. Includes --no-force and --no-vision. Kinematic checks are applied, but the arm is not moved. This option (together with --no-mobile) can be used to run robotd in a simulator mode.

--no-mobile disables the mobile platform. No connection to the mobiled is opened, no checks are done and the mobile displacement is silently be assumed to be zero. This option (together with --fake-arm) can be used to run robotd in a simulator mode.

--no-force disables the force sensor. All force are assumed to be zero. This option is only useful for debugging or experimental setups because it breaks several operations.

--no-vision disables the vision system. No connection is made to the image server, no computations done, all classifications are silently assumed to succeed and all displacements to be zero.

6.6.3 Modifying/Extending the Functionality

In compliance with the level of flexibility required by the task definition the functionality implemented with `CROBOT` and `CROBOTD` is not static, but can be changed to a large degree. Adaptions of the robot to changes of the environment or the biotechnological process can therefore be easily performed. Depending on the scope of the changes they can even be possible at run-time without restarting the system, which is a major advantage in a setup with untrained personnel because it allows a remote administration.

The possible changes can be divided into four groups of increasing complexity and decreased simplicity:

1. Modifying existing scripts/parameters:

   If only a position has to be changed (for example the target relative to the vision marker has changed) or an existing script has to be modified (for example a motion has to be added) this can be most easily accomplished. These type of changes can be made in the INFOBASE at run-time and do not require to restart the `robotd` or the laboratory control program. With a typical laboratory use the vast majority of changes fall into this category.

2. Adding new scripts:

   If a new script has to be added to perform some functionality that is not yet included in any other script this requires slightly more work. Basically, the script has to be added to the INFOBASE and a proper method has to be added to `CROBOT`. This function must set up a `CLABMSG`, that is, fill its fields

   ```
   CLABMSG._cmd = CMD_EXEC_SCRIPT
   CLABMSG._stringData[0] = strdup ("<scriptname>")
   ```

   just as all other methods do. Since the script name is transferred as a string and not some magic value that has to be handled in a case-switch this does not require to restart the `robotd`, although it does require to restart the laboratory control program (or whatever program is sending the commands). With a typical laboratory use these changes are unlikely to be necessary.

3. Adding new devices:

   If a new device has to be added (for example an extra sampling device) the necessary changes depend on the device in question. As long as the device can be operated with the existing primitives all that supporting it involves is adding a new script/transforms as described above. If the device can not be operated with the provided primitives the `CROBOTD` class has to be extended by the appropriate methods, as described below. These changes, however, mostly occur during initial installation of the system only and have no impact on the typical use.

4. Adding new motion primitives:

   If a new motion primitive has to be added (for example a new type of force controlled motion) the `CROBOTD` class has to be extended by the appropriate methods. Depending on the primitive in question this will involve more or less explicit knowledge about C++ programming in general and RCCL in particular. It will require an experience programmer and is therefore the most complex enhancement, but on the other hand it has not yet been necessary after the initial establishment of the system.

Since the modifications which involve C++ programming are not really necessary during typical use and the easier ones only involve changing the INFOBASE the system can therefore be considered as easily maintainable.

6.7 Summary

A task as complex as *automating the sample management* can not be done without requiring any knowledge from the operator – it plainly and simply is a lot more complex than programming a VCR. It also must not be forgotten that the project was supposed to be a *case study*, not to result in a *product*. Because of this, those parts of the system that may/have to be edited/configured by the operator still require knowledge about programming and robots, though the complexity has been brought down to a reasonable level by various techniques. This section summaries the efforts and gives an outlook into possible future enhancements.

Scripts

The robot actions in this work are controlled by scripts. The scripts contain simple commands to make use of the vision system, to set motion parameters, to move the robot according to position equations as well as a few special motion types and to do limited conditional branching. The scripts may seem complex for a layperson to deal with, however – upon a closer look – the obvious alternatives are hardly less complex.

Using *teaching* for example appears like an interesting idea. Instead of programming the motions using position equations a special “zero gravity mode” of the arm can be used to passively move it. Motions to targets which are not reachable on a straight line will not have to be split up into several parts because the operator will “show” the system a proper *trajectory*, and not just program the *target*. However, one drawback with this approach is that there are cases where precision and/or straight line motions are needed, both of which is not possible with teaching.

Another idea is to use the zerograv mode, a joystick, mouse or graphical interface to move the robot to store only a few setpoints through which a trajectory is then interpolated, for example by means of straight lines or splines. This approach can provide the necessary precision and use smooth and time-optimal trajectories. However, like direct teaching, it has the disadvantage that with no knowledge about the situation the system can do nothing but store exactly the learnt trajectory and apply it to exactly the same situation again. In case of devices with several buttons the operator can take no benefit from knowing that one button is a certain distance away from another – he/she will have to teach a completely separate trajectory for every button. This also applies if a task change should make it necessary to press other buttons, or to press buttons in a different order. More basically, trajectories from all possible start points to all possible targets have to be teached, and this is hardly a simplification.

The very major drawback with all these ideas is that none of them will completely rid the operator of learning/using a script language because specifying the targets is only one part of the problem. Accessing sensors and doing conditional motions is the other part, and any means of getting rid of position equations does not help here. Therefore the implemented script functionality can be seen as a reasonable compromise.

State Automaton

A DFA is also used to provide a compromise between effort and safety. It catches the only problem caused by accidental mis*usage* of the system by personnel that has been encountered so far – the mixing up of the command order. All other problems were caused by the operator *programming* the scripts, which is something that will not happen during typical use. Extending the checks to also catch pitfalls during the programming state is not really possible. The problem is that a proper balance between helping the programmer and restricting the programmer has to be found.

All programming languages only check for syntactical errors, not for semantical errors. The reason is simply that they do not see the semantic because it is not expressed in the language. In case of a programming language the system does not know what a routine is supposed to do, and therefore it can not help. In case of a robot setup the system does not know how to perform a task, and therefore it can not help, too.

The plain syntax errors can be checked by running the `robotd` in simulator mode. This also checks if all required transforms are defined and if no motion targets – assuming no device displacements – are out of reach. All further checks like collision checks require a motion simulator in RCCL, which is not available for the PA-10 arm.

Also important is the fact that most problems can not be dealt with by offline simulation because they arise online from the device displacements that have to be detected and compensated by the vision. The only thing that could be done is computing average and assuming worst case displacements and see how they affect the reachability or singularities. Expected singularities are dealt with by a special mode which does velocity clipping, but since this deviates from the desired trajectory it may only be used when not in contact with any object[18]. Predicting the displacement is something the system can not do and so it is the operator's responsibility to design the situation so that there is enough safety margin for the worst case displacement.

Therefore no additional checks at all are implemented and the state automaton has to be seen as a reasonable compromise. The fact that it is a *deterministic* automaton whereas the system itself is *non-deterministic* because of external influence can not be avoided because no sensors exist to monitor the external state changes. It is not within the scope of the state automaton to address these effects.

Database

The INFOBASE used to keep parameters and scripts also has to be seen as a compromise between functionality on the one hand side and portability and complexity on the other hand side. It is similar to a very simple networked filesystem, but in its simplicity lies one of its great advantage: It can be easily ported to any operating system. While commercial databases are network-capable too, they do not offer network client frontends for all operating system, in particular not the QNX operating system used for some computers in this work. LDAP may be an alternative for a rewrite, but was not yet commonly known at the time of the system design.

Another great advantage also lies in its simplicity: Because of its focus on the specific requirements of this work it is very lean and easy to use. Both the INFOBASE itself as well as the

[18] See [Scherer 1998].

client frontend CINFOBASE require only very few resources and can easily be used on small computers, like embedded controllers on intelligent devices.

Since the INFOBASE is found by clients using broadcasts it also serves as a lookup mechanism for laboratory services like the MFCS or the robot. Services register the IP address and port number they are running on under a unique name in the INFOBASE. This means that a client need not know the host a service is running on, but only its name. It can look up the address using the name in the INFOBASE. This mechanism works as long as all involved devices are within the same subnet and can receive broadcasts.

The worst that can be said about the INFOBASE is that it represents a *single point of failure* – without the INFOBASE nothing will work – but here its simplicity has helped to practically prove its stability over the years. It is a central component of the system and well tailored to its requirements.

Network Support

Programming distributed applications with network communication involves a lot of details one usually does not want to bother with. Programming languages like Java come with extensive libraries that relieve the user from a large part of this burden, however, C/C++ do not include such standardised tools, and so a non-standard tool had to be used. The CNETOBJ and CLABDEV mechanisms implemented for this provide C++ classes for encapsulation of the object network transmission and the one-shot client/server communication that is required by the system. Like the other parts, they focus on providing the necessary service with the least possible overhead and complexity.

As such, they are less powerful than other network communication tools like CORBA, but the same argument as with the INFOBASE applies: Having a lean, well tailored and yet cheap solution is in most cases advantageous compared to complex and omnipotent products. CNETOBJ and CLABDEV therefore implement a suitable approach to minimising the network details required to be known to use the distributed robot system. They are used for many services within the system with no known problem.

The Robot

With the above preparations, controlling the robot is only a matter of consequential usage of these tools. The C++ class CROBOT used on the client side completely hides all network and robot issues from the user and does therefore not require special knowledge. The C++ class CROBOTD used on the server side does require special knowledge about robots, but concerns only the programmer initially establishing the system, and not the final user. The final user can still determine the sequence by choosing the methods of the CROBOT class to be called – as long as the sequence is allowed by the state automaton – and by modifying their parameters in the INFOBASE – both of which are trivial operations. This way of controlling the robot has proven itself successful during the entire project.

Chapter 7

Automated Sample Management Results

"Six pints of bitter. And quickly please, the world's about to end."

Ford Prefect

Usually, a technical system needs to pass several tests with good results, and usually, these results are presented in a separate, but single chapter. Results from the individual components of the mobile robot system presented in this work have so far been given in section 3.6 for the mobile platform, section 4.4 for the robot arm and section 5.5 for the vision system. According to these results the system should clearly be able to perform the automated sample management according to the task definition in section 2.3, however, they do not actually prove that it does. With a system as complex as this one, that employs several unpredictable sensor information, such a proof can not formally be given. Therefore, the only way to demonstrate the system's fitness is to do additional test with its intended application, the automated sample management. For a feasibility study one successful test would already be enough, but for this system – apart from numerous dry runs – several real test cultivations have been done using the mobile robot system, and their results are given in this chapter.

The results given consist of descriptions and photos taken during one of the test cultivations in section 7.1 to show how the robot performs its actions, and the biotechnological results in section 7.2 to show that the automated sample management has no effect on the quality of the sample management.

7.1 Automated Sample Management

In the dry runs and test cultivations the robot actions were controlled by the central laboratory control program (LCP) by I. Poggendorf accessing all the auxiliary devices too and creating the protocol files[1]. The LCP uses the `CROBOT` interface from section D.4 in the appendices to call methods from table 7.1 to perform the sample management. The sequence of actions it performs is not fixed, and so the run given in this section is only one example of a meaningful sequence. It does in particular not make use of the *entire* set of methods available in `CROBOT`, which is why table 7.2 lists even more methods.

One reason why the LCP does not use them is that they separately implement atomic actions, which were designed to be used in the first tests only. The entire sequence can be done with

[1] See [Poggendorf 2004].

`UnparkCharger()`	moves away from the charging station, ready for a new run.
`PickCedexCedex()`	moves to the cedex (whose carousel must be in the proper position) and picks the cedex tube from the slot.
`PlaceCedexWaste()`	moves to the waste bin and disposes the cedex tube.
`PickTubeStorage()`	moves to the storage rack for normal $50\,ml$ NUNC tubes and grasps one from a full slot.
`HoldTubeSampler()`	moves to the sampler, places a tube into the rendezvous slot and fixates it so that the sampler may run. May only be followed by `TakeTubeSampler()`.
`TakeTubeSampler()`	following `HoldTubeSampler()`, takes the tube out of the sampler again.
`HoldTubePipette()`	moves to the pipette, optionally shakes the tube and holds it under the needle. May only be followed by `TakeTubePipette`.
`TakeTubePipette()`	following `HoldTubePipette`, takes the tube out of the pipette again.
`LoadAndRunCentrifuge()`	time-saving combination: moves to the centrifuge (which must be open), places the tube in the cage (optionally rotating it), closes the lid (really) and presses the "start" button.
`PickCedexStorage()`	moves to the cedex tube storage and picks a cedex tube from a slot.
`HoldCedexPipette()`	moves to the pipette and holds the cedex tube under the needle, staying there. May only be followed by `TakeCedexPipette`.
`TakeCedexPipette()`	takes the cedex tube away from the pipette.
`PlaceCedexCedex()`	moves to the cedex (whose carousel must be in the proper position) and places the cedex tube in the slot.
`OpenFridge()`	moves to the fridge and opens the sliding lid.
`StopAndUnloadCentrifuge()`	time saving combination: moves to the centrifuge (which must be closed but idle), presses the "lid" button, opens the lid and picks the tube from the cage (optionally rotating it).
`HoldTubePipette()`	moves to the pipette, optionally shakes the tube and holds it under the needle. May only be followed by `TakeTubePipette`.
`TakeTubePipette()`	following `HoldTubePipette`, takes the tube out of the pipette again.
`PlaceTubeWaste()`	moves to the waste bin and disposes the currently grasped tube.
`PickTubeStorageBarcode()`	same as `PickTubeStorage`, only that the storage rack for barcode labelled tubes is used.
`HoldTubePipette()`	moves to the pipette, optionally shakes the tube and holds it under the needle. May only be followed by `TakeTubePipette`.
`TakeTubePipette()`	following `HoldTubePipette`, takes the tube out of the pipette again.
`HoldTubeScanner()`	moves to the barcode scanner and holds a tube in front of it. May only be followed by `RotateTubeScanner` or `TakeTubeScanner`.
`RotateTubeScanner()`	rotates the tube in front of the scanner by a small random angle around the world z-axis.
`TakeTubeScanner()`	takes a tube away from the scanner
`PlaceTubeAndCloseFridge()`	time-saving combination: moves to the fridge (which must be open), places the tube in a free slot and closes the sliding lid.
`ParkCharger()`	moves to the charging station and parks there, recharging batteries. May only be followed by `UnparkCharger`

Table 7.1: Robot methods implemented by `CROBOT` and used by the laboratory control program for the automated sample management.

these atomic actions, but they waste a lot of time by going back into the park position after each command. Since time is an important – though biotechnologically not that critical – constraint, time-saving combinations of them have been implemented and are now preferred. However, since the atomic actions are partially based on the same scripts it does not hurt to keep them for possible future use in a modified setup.

`PlaceTubeStorage()`	moves to the storage rack for normal $50\,ml$ NUNC tubes and puts the currently grasped tube back into an empty slot.
`PlaceTubeStorageBarcode()`	same as `PlaceTubeStorage`, only that the storage rack for barcode labelled tubes is used.
`OpenCentrifuge()`	moves to the centrifuge, presses the "lid" button and opens the lid.
`PlaceTubeCentrifuge()`	moves to the centrifuge (which must be open) and places the tube into the cage.
`PickTubeCentrifuge()`	moves to the centrifuge (which must be open) and picks the tube from the cage.
`RunCentrifuge()`	moves to the centrifuge and presses the "start" button.
`CloseCentrifuge()`	moves to the centrifuge and closes the lid.
`CloseCentrifugeReally()`	moves to the centrifuge and pushes down an already closed lid where the locking mechanism has not snapped in.
`PlaceTubeFridge()`	moves to the fridge (which must be open) and places the tube in a free slot.
`CloseFridge()`	moves to the fridge (which must be open) and closes the sliding lid.

Table 7.2: More robot methods implemented by `CROBOT` but not actually used by the laboratory control program.

7.1.1 The Sample Management Sequence

In this subsection the sequence of actions of the LCP according to a sample log file in table 7.3 will be discussed, with the focus on putting the robot actions into the proper context. The individual robot actions will then be explained further in the next subsection.

The steps according to the log of the LCP are:

- ROBOTERCHARGERWEG calls

  `UnpackCharger()` to undock the mobile robot from its charging station and make it ready for the new cycle.

- VORBEREITENCEDEX logs into the Cedex via TCP/IP and reserves it to be used by the robot, temporarily blocking console access.

- ALTESCUPCEDEX commands the Cedex (AS20) to eject the old Cedex tube and to turn the carousel so that the robot can grasp it[2]. After this, it calls

  `PickCedexCedex()` to move the mobile robot to the Cedex and grasp the Cedex tube from the slot.

- ROBOTERCEDEXABFALL calls

  `PlaceCedexWaste()` to move the mobile robot to the waste bin and drop the Cedex tube into it.

  The steps until here are only done to initialise the system and to clean up from the previous cycle, but now the new cycle starts.

- ROBOTERVORRATPROBE calls

  `PickTubeStorage()` to move the mobile robot to the storage rack for NUNC tubes without barcode and grasp a new tube.

[2]The old tube is left in the Cedex after a measurement because the Cedex needs a tube to flush its syringe and it does not hurt to leave it there after completition of the cycle.

Mon Mar 24 22:24:01 2003
****
Die Befehlssequenz beginnt jetzt...
****

RoboterChargerWeg	Mon Mar 24 22:24:01 2003		Mon Mar 24 22:24:08 2003
VorbereitenCedex	Mon Mar 24 22:24:08 2003	rob030305 rob030305 R018	Mon Mar 24 22:24:32 2003
AltesCupCedex	Mon Mar 24 22:24:32 2003		Mon Mar 24 22:24:32 2003
RoboterCedexAbfall	Mon Mar 24 22:24:32 2003		Mon Mar 24 22:26:34 2003
RoboterVorratProbe	Mon Mar 24 22:26:34 2003		Mon Mar 24 22:27:42 2003
RoboterReaktorWarten	Mon Mar 24 22:27:42 2003		Mon Mar 24 22:28:51 2003
Reaktor1Entnahme	Mon Mar 24 22:28:51 2003		Mon Mar 24 22:30:26 2003
Wartezeit	Mon Mar 24 22:30:26 2003	280	Mon Mar 24 22:35:06 2003
RoboterReaktorPipette	Mon Mar 24 22:35:06 2003		Mon Mar 24 22:36:45 2003
PipettierenProbe	Mon Mar 24 22:36:45 2003	Spuelen1	Mon Mar 24 22:38:31 2003
RoboterPipetteWeg	Mon Mar 24 22:38:31 2003		Mon Mar 24 22:38:51 2003
RoboterLadenZentrifuge	Mon Mar 24 22:38:52 2003		Mon Mar 24 22:40:53 2003
RoboterCedexPipette	Mon Mar 24 22:40:53 2003		Mon Mar 24 22:42:25 2003
AusgebenProbe	Mon Mar 24 22:42:25 2003	1.03	Mon Mar 24 22:42:40 2003
RoboterPipetteCedex	Mon Mar 24 22:42:40 2003		Mon Mar 24 22:44:08 2003
StartenCedex	Mon Mar 24 22:44:08 2003	rob030305	Mon Mar 24 22:44:36 2003
RoboterOeffnenTiefkuehl	Mon Mar 24 22:44:37 2003		Mon Mar 24 22:46:13 2003
PipetteSpuelen2	Mon Mar 24 22:46:13 2003		Mon Mar 24 22:51:38 2003
FertigZentrifuge	Mon Mar 24 22:51:38 2003	Zeit	Mon Mar 24 22:51:38 2003
RoboterZentrifugePipette	Mon Mar 24 22:51:38 2003		Mon Mar 24 22:54:51 2003
PipettierenArchiv	Mon Mar 24 22:54:51 2003	2.5	Mon Mar 24 22:55:13 2003
RoboterPipetteAbfall	Mon Mar 24 22:55:13 2003		Mon Mar 24 22:56:27 2003
RoboterArchivPipette	Mon Mar 24 22:56:27 2003		Mon Mar 24 22:58:26 2003
AusgebenArchiv	Mon Mar 24 22:58:26 2003	2.5	Mon Mar 24 22:58:40 2003
RoboterArchivBarcode	Mon Mar 24 22:58:40 2003		Mon Mar 24 22:59:20 2003
StartenBarcode	Mon Mar 24 22:59:20 2003	0226	Mon Mar 24 22:59:23 2003
RoboterBarcodeWeg	Mon Mar 24 22:59:23 2003		Mon Mar 24 22:59:34 2003
RoboterTiefkuehlUndZu	Mon Mar 24 22:59:34 2003		Mon Mar 24 23:01:46 2003
ErgebnisCedex	Mon Mar 24 23:01:46 2003	Probe:R018 Tot:2.17E+005 Lebend:9.41E+005 Viability:81.3 SD:1.42E+005 SDrel:15.09 Bilder:20	Mon Mar 24 23:01:59 2003
ErgebnisMFCS	Mon Mar 24 23:01:59 2003	Tot:2.17E+005 Lebend:9.41E+005 Viability:81.3 SD:1.42E+005 SDrel:15.09 Bilder:20 Barcode:226 Cedex:18	Mon Mar 24 23:02:04 2003
RoboterChargerHin	Mon Mar 24 23:02:04 2003		Mon Mar 24 23:02:39 2003
PipetteSpuelen2	Mon Mar 24 23:02:39 2003		Mon Mar 24 23:08:05 2003

Mon Mar 24 23:08:05 2003

****
Die Befehlssequenz wurde vollstaendig abgearbeitet!
****

Table 7.3: A sample log file of a run of the sample management as printed by the laboratory control program by I. Poggendorf.

- ROBOTERREAKTORWARTEN calls

  `HoldTubeSampler()` to move the mobile robot to the sampling device, place the tube in the slot and fixate it.

- REAKTOR1ENTNAHME uses the sampling device to first flush the tubes with cell broth, then penetrate the needle through the septum of the tube, fill the sample and retreat the needle again. The entire control of the sampling device is fully autonomous, so that the remaining system can continue to do other tasks while it is working in the background. Since the robot has to fixate the tube this feature is not used.

- WARTEZEIT waits for a calibrated amount of time after which it is safe to take the tube out of the sampler again.

- ROBOTERREAKTORPIPETTE calls

  `TakeTubeSampler()` to take the tube from the sampler again and

  `HoldTubePipette()` to move the mobile robot to the pipette and hold the tube under its needle.

- PIPETTIERENPROBE accesses the pipette to suck in 1 ml of the cell broth.

- ROBOTERPIPETTEWEG calls

  `TakeTubePipette()` to take the tube from the pipette again.

- ROBOTERLADENZENTRIFUGE calls

  `LoadAndRunCentrifuge()` to move the mobile robot to the centrifuge, optionally rotate the cage so that the free slot is in a proper position, place the tube in the free slot, close the lid of the centrifuge and press its start button. The centrifuge is then left running in the background.

- ROBOTERCEDEXPIPETTE calls – while the centrifuge is running –

  `PickCedexStorage()` to move the mobile robot to the storage racks for Cedex tubes and grasp a new tube and

  `HoldCedexPipette()` to move the mobile robot to the pipette and hold the Cedex tube under its needle.

- AUSGEBENPROBE then ejects the 1 ml cell broth into the Cedex tube.

- ROBOTERPIPETTECEDEX calls

  `TakeCedexPipette()` to take the Cedex tube from the pipette again and

  `PlaceCedexCedex()` to move the mobile platform to the Cedex and place the Cedex tube into the slot in the carousel.

- STARTENCEDEX starts the Cedex counting process which keeps running in the background.

- ROBOTEROEFFNENTIEFKUEHL calls

  `OpenFridge()` to move the mobile robot to the fridge and open its sliding lid. Since the lid is on top of the fridge the cold is not lost by leaving it open for a few minutes.

- PIPETTESPUELEN2 performs the cleaning process of the pipette. This process does not require a tube to be under the needle.

- FERTIGZENTRIFUGE waits until the centrifuge reports that it is ready, which is usually immediately the case because the steps in between have taken a few minutes time.

- ROBOTERZENTRIFUGEPIPETTE calls

`StopAndUnloadCentrifuge()` to move the mobile robot to the centrifuge, press the button to unlock the lid, open the lid, optionally rotate the cage so that the slot with the NUNC tube is in a proper position and grasps the tube and

`HoldTubePipette()` to move the mobile robot to the pipette and hold the tube under its needle.

- PIPETTIERENARCHIV accesses the pipette to suck in 2.5 ml of the cell free supernatant of the broth.
- ROBOTERPIPETTEABFALL calls

  `TakeTubePipette()` to take the NUNC tube from the pipette again and

  `PlaceTubeWaste()` to move the mobile robot to the waste bin and drop the NUNC tube into it.

- ROBOTERARCHIVPIPETTE calls

  `PickTubeStorageBarcode()` to move the mobile robot to the storage rack for NUNC tubes with barcode and grasp a new tube and

  `HoldTubePipette()` to move the mobile robot to the pipette and hold the tube under the needle.

- AUSGEBENARCHIV ejects the 2.5 ml cell free supernatant into the tube.
- ROBOTERARCHIVBARCODE calls

  `TakeTubePipette()` to take the tube from the pipette again and

  `HoldTubeScanner()` to move the mobile platform to the barcode scanner and hold the barcode in front of the laser.

  It then accesses the barcode scanner to read the barcode. If this fails because the scanner can not see the barcode it calls

  `RotateTubeScanner()` to rotate the tube in front of the barcode scanner by a small, random angle.

  It repeats this until the barcode can be read.

- ROBOTERBARCODEWEG calls

  `TakeTubeScanner()` to take the tube from the barcode scanner again.

- ROBOTERTIEFKUEHLUNDZU calls

  `PlaceTubeAndCloseFridge()` to move the mobile robot to the fridge, place the NUNC tube in a free slot inside it and close the sliding lid.

- ERGEBNISCEDEX collects the results from the Cedex and logs off it to allow it to be used by other users again.

- ERGEBNISMFCS transfers all results to the MFCS.
- ROBOTERCHARGERHIN calls

  `ParkCharger()` to move the mobile robot to the charging station and dock to it.

- PIPETTESPUELEN2 is again a cleaning cycle of the pipette.

After the final flushing and cleaning of the pipette the cycle is finished and the LCP waits for a new cycle to begin.

7.1.2 The Robot Actions

The robot actions performed in this cycle require a more detailed presentation. They will therefore be discussed now and some of them illustrated with force logs and images.Robot actions which are used more than once during the cycle are mentioned only once in the following, so the sequence given here does no longer correspond to a meaningful sample management cycle.

UnparkCharger()

Following a previous `ParkCharger()` this method undocks the mobile platform from its charging station. In order to do this it uses a relative backward motion during which the collision avoidance is switched off, making it a bit troublesome. The reason for switching off the collision avoidance is that the charging station is mounted to a pillar which the collision avoidance would otherwise recognise as an obstacle and halt the motion. Currently the collision avoidance is disabled for both laser scanners, but this can be fixed to only disable the front scanner facing the pillar and thus still allow collision avoidance in the direction of the motion.

PickCedexCedex()

This method is the first that uses the regular sequence of primitives: It first moves the mobile platform to the cedex, positions the camera over the expected position of the marker by moving the arm, uses the vision system to classify the marker and compensate its displacement, then positions the camera over the expected position of the slot (in this case it is in fact a second marker on the carousel close to the slot) relative to the initial marker, compensates its displacement too, grasps the Cedex tube and retreats the arm into the park position. During picking the tube the arm first moves down until it detects a deliberate collision with the carousel and then back up a millimetre again.

PlaceCedexWaste()

Dropping a Cedex tube into the waste bin is trivial and yet special because the waste bin currently does not have a marker for classification and fine-positioning. The sequence of primitives is therefore reduced to moving the mobile platform to the waste bin, moving the arm to the expected position, simply open the gripper and drop the tube into whatever is under the gripper and retreating to the park position. Apart from not being guaranteed to succeed this has so far been considered sufficient because the waste bin is so large that no fine-positioning is needed.

PickTubeStorage()

This method moves the mobile platform to the storage rack for NUNC tubes without barcode, uses the vision system to classify the rack and compensate its displacement, selects a filled slot based on status information in the INFOBASE, centres over the tube, grasps it and retreats to the park position. Figure 7.1 illustrates this sequence. During grasping the gripper is first moved down until a deliberate collision with the rack is detected by the force sensor, and then back up a calibrated distance to the cup of the tube. This way a limited variation of the height of the storage rack can be compensated.

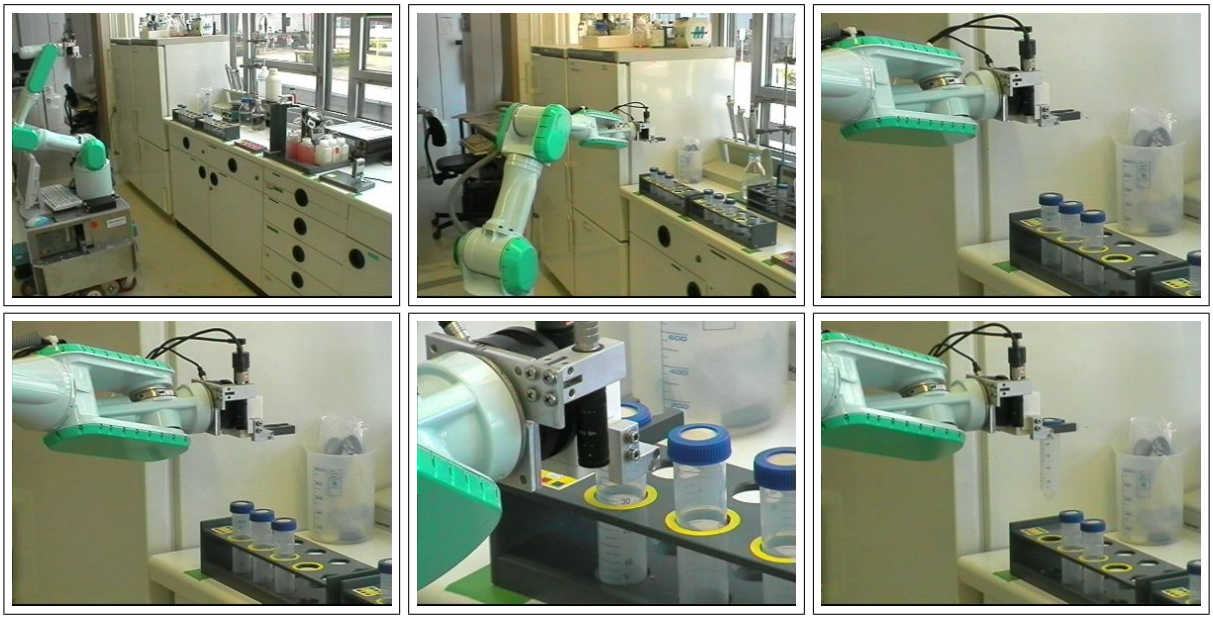


Figure 7.1: PickTubeStorage()

HoldTubeSampler()

The next step is to move the mobile platform to (one of the) sampling station(s), perform the usual steps of classification and fine-positioning and place the tube in the free slot as in figure 7.2. Selection of the free slot is done using the same mechanism as with the storage racks, however, only one slot is actually stored in the INFOBASE. This means that always the same slot is chosen, but that this slot can be changed rather easily[3].

After placing the tube in the slot the arm repositions itself to regrasp the tube below the cap to fixate it. This is needed because the sampler – when removing the needle from the tube – will otherwise pull the tube out of the slot because the needle will get stuck in the septum sealing the tube. The arm remains holding the tube after completition of this method. It may only be followed by `TakeTubeSampler()`.

During placement of the tube in the slot the force sensor is used to detect whether the tube got stuck, but not to implement a real force controlled peg-in-hole operation. During regrasping it is used too to detect the height of the storage rack just like in `PickTubeStorage()`.

[3]Different coordinates for the slots relative to the marker are in fact the only change that is needed for the three different sampling devices.

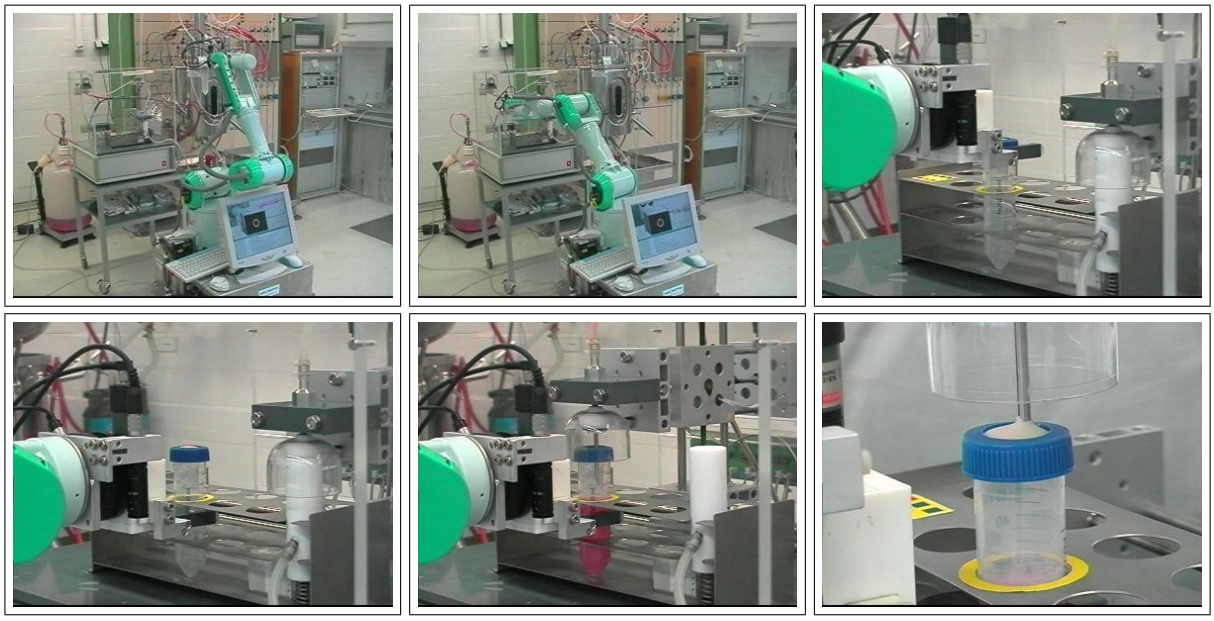


Figure 7.2: HoldTubeSampler()

TakeTubeSampler()

After `HoldTubeSampler()` regrasped the tube to fixate it during sampler operation `HoldTubeSampler()` now reverses this sequence and grasps the tube in the normal way again as in figure 7.3. Since the arm has been in contact with the tube in the meantime no additional fine-positioning is needed for this. The tube is then taken out of the slot and the sampler and the arm moved back into a park position.

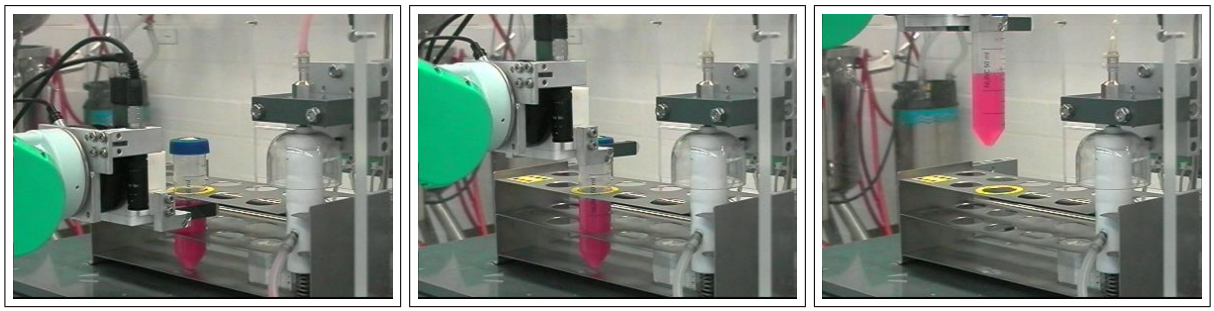


Figure 7.3: TakeTubeSampler()

HoldTubePipette()

This method moves the mobile platform to the pipetting station, performs the usual steps of classification and fine-positioning and holds the tube under the needle of the pipette as in figure 7.4. The arm remains holding the tube after completition of this method. It may only be followed by `TakeTubePipette()`.

Figure 7.4: HoldTubePipette()

The depth how far the needle penetrates into the tube is given as an additional parameter. The force sensor is not used during the penetration of the needle because the forces the needle exerts on the septum of the tube are too small to be separated from the basic noise.

TakeTubePipette()

Following `HoldTubePipette()` this method takes the tube out of the pipette again and moves the arm back into its park position. Since all relevant positions have been stored no vision or force sensor is needed during this operation.

LoadAndRunCentrifuge()

Instead of using multiple separate commands to operate the centrifuge, the entire first half of operations concerning it has been integrated into `LoadAndRunCentrifuge()`. This method moves the mobile platform to the centrifuge, classifies it and fine-positions over the coloured buttons on the front panel (the centrifuge is the only device which is not labelled with a marker), then centres over the cage inside the centrifuge, detects the position of the free slot, optionally rotates the cage so that the slot is in a proper position, inserts the tube into the slow, retreats the arm to move it behind the lid, closes the lid with a circular motion downwards, again repositions the arm to try and lift the lid again to see if it locked close and press onto it to really close it if it is not and finally presses the “START” button to start the centrifuge and moves the arm back into its park position. A visualisation of this sequence is given in figure 7.5. The centrifuge is left running after this method returns.

The arm may have to rotate the cage because it can not reach the slot in all possible positions of the cage and the centrifuge does not stop in a defined position[4]. In these cases (the script conditionally branches here) it applies a series of rotations to the cage until the slot is in a proper position ($\leq \pm 45$ degrees from the point where the arm is standing) and only then grasps the tube. To allow it to perform the rotations it must temporarily deposit the tube in a small storage rack mounted to the centrifuge, and later grasp it again. The rotation is then applied by pushing the gripper onto the surface of a pad on the cage and using “telephone-dialing” like circular motions. During these circular motions the force with which the gripper presses onto the pad is controlled because it must not loose contact.

During insertion of the tube into the slot the force sensor is used to implement a force controlled peg-in-hole operation and to detect when it is fully inserted. It is also used during testing

[4]There exist centrifuges that do stop in a defined position, but they are way more special and expensive. They are not used for this project because the intention was to use standard (and cheap) equipment.

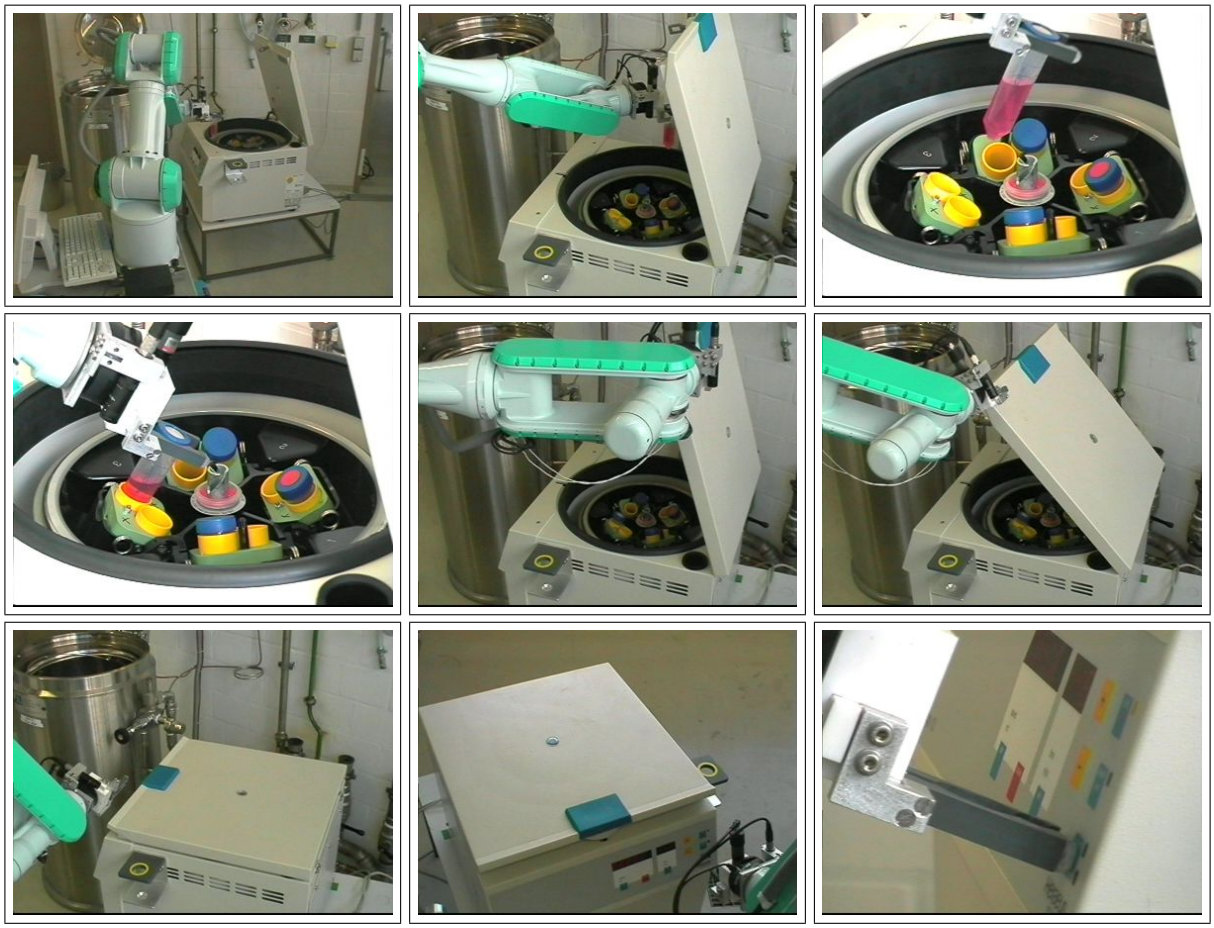


Figure 7.5: LoadAndRunCentrifuge()

whether the lid is really closed or not. If the lid is not closed (the script again conditionally branches here) the arm is positioned over it and starts a motion to slowly press it down against the rubber sealing with an increasing force. As can be seen in figure 7.6 at the moment when the locking mechanism snaps in a spike occurs in the force, which it detected by the force sensor and used to abort the motion. The force of 60 to 70 N at which this usually occurs is more than 20 times the force that is otherwise used during force control or collision detection and more than half of the maximum force of 110 N that the sensor is designed for.

PickCedexStorage()

This method moves the mobile platform to the storage rack for cedex tubes, does the usual classification and fine-positioning, selects a slot, centres over the slot, grasps a new Cedex tube and goes back into the park position. Figure 7.7 shows a visualisation of this sequence, which really is the same as with `PickTubeStorage()`, only that the distances differ and thus the lengths of the relative motions.

HoldCedexPipette()

Similar to `HoldTubePipette()`, this method moves the mobile platform to the pipetting station, classifies it and fine-positions over its marker and performs the necessary relative motions to hold the Cedex tube under the needle. Contrary to `HoldTubePipette()` this happens at

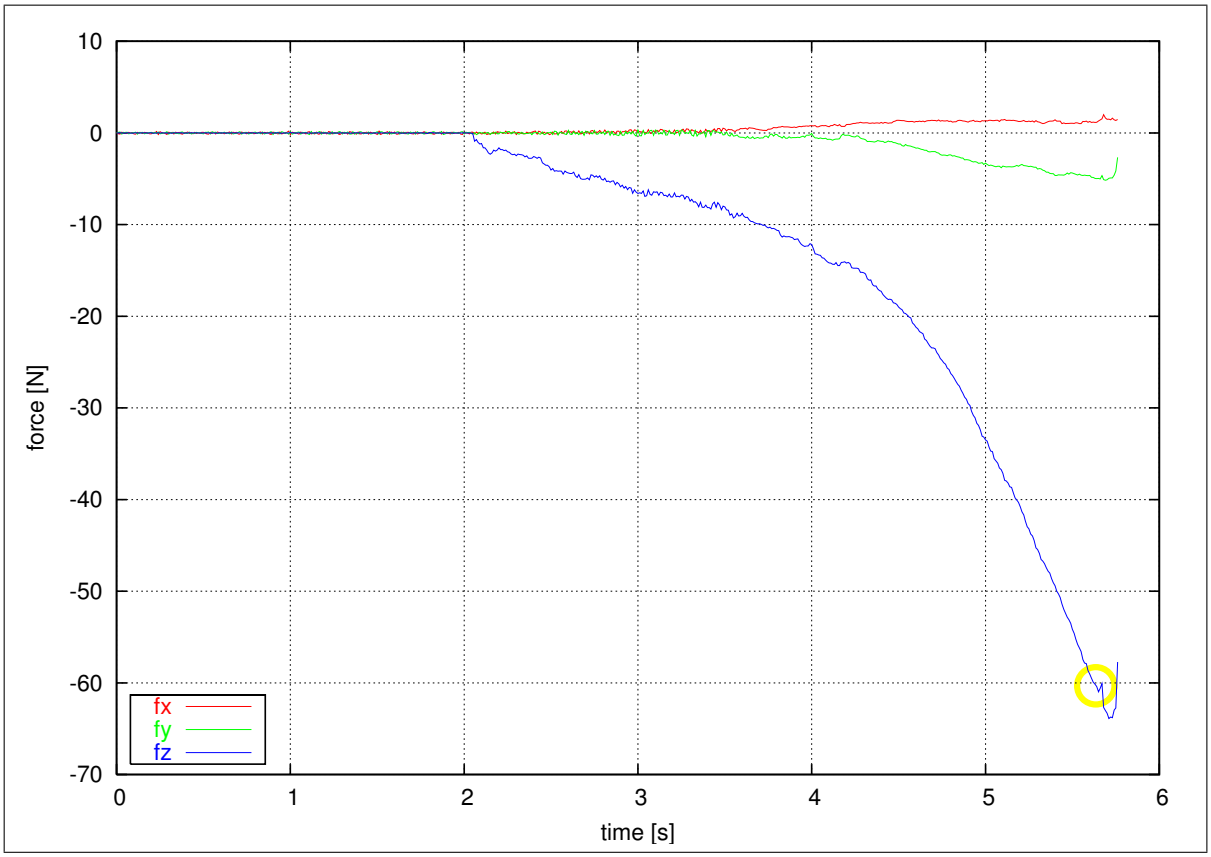


Figure 7.6: LoadAndRunCentrifuge() – Forces during closing the lid. The small spike marked with a yellow circle is the point where the locking mechanism snaps in.

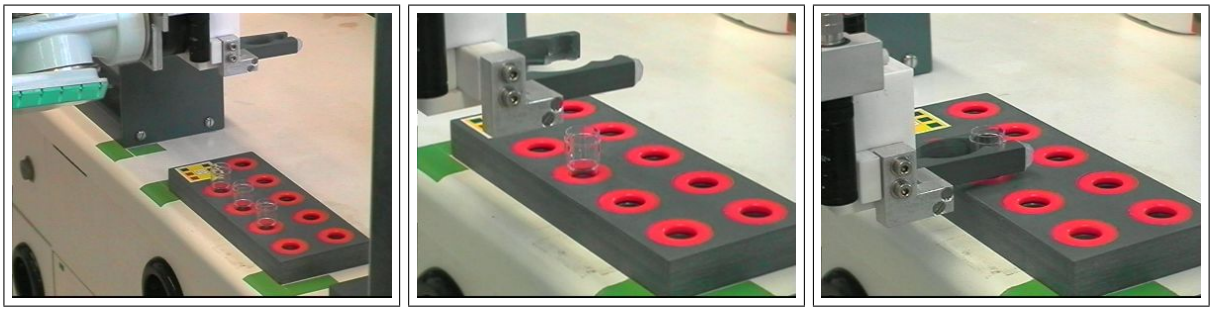


Figure 7.7: PickCedexStorage()

a different height because the Cedex tube is much smaller. Since it is not necessary to control the penetration depth of the needle the arm is moved upwards until the force sensor detects that the needle has reached the bottom of the tube. The arm is then moved 2.5 mm down again to not block the hole in the needle. This way the length or height of the needle can be varied in a limited range without having to modify the script. Figure 7.8 shows the steps of this sequence. The arm is left holding the Cedex tube after this method returns. It may only be followed by `TakeCedexPipette()`.

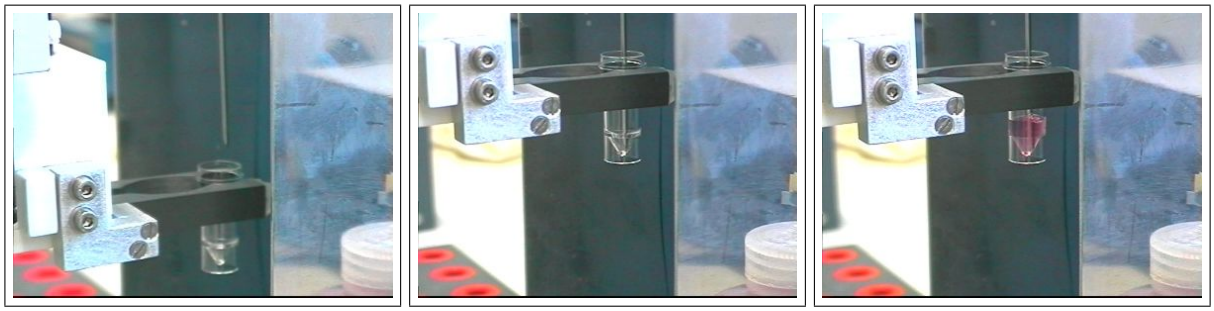


Figure 7.8: HoldCedexPipette()

TakeCedexPipette()

Following `HoldTubePipette()` this method removes the Cedex tube from the needle of the pipette by reversing the previous motion sequence and moving the arm back into its park position. Since all relevant positions have been stored no vision or force sensor is needed during this operation.

PlaceCedexCedex()

This method drives the mobile platform to the Cedex cell counter, does the usual classification and fine-positioning over the marker on its carousel, then positions over the frontmost slot, inserts a Cedex tube into it and goes back into the park position. During the insertion of the tube the force sensor is used to detect whether it really hits the slot or collides with the carousel. Figure 7.9 illustrates this sequence.

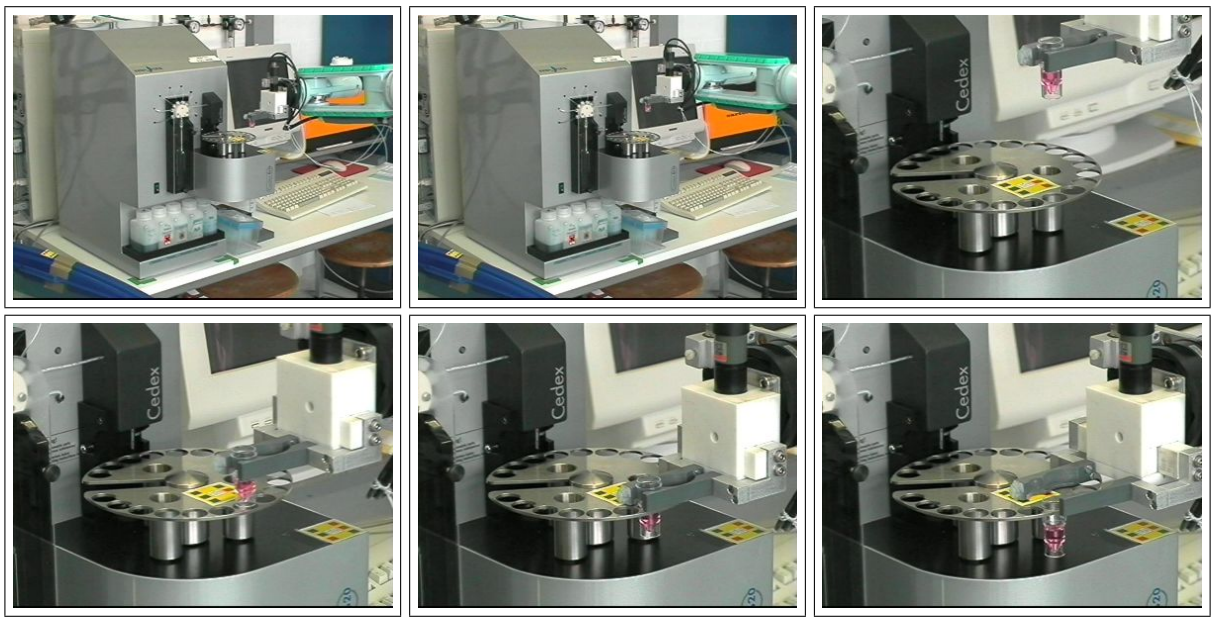


Figure 7.9: PlaceCedexCedex()

The slot is actually not found by centring over *it*, but by centring over a second marker that is attached to the carousel as close to the slot as possible. This is because contrary to all other cases involving slots the dimensions of the carousel are such that it is not possible to add a

coloured ring around the slot. Since a symmetric object like a ring only allows to centre over its *position*, but not to adjust to its *orientation*, placing a ring next to the slot is not feasible unless the direction from the ring to the slot can be given otherwise (which it can not in this case). Only an additional marker can also give the direction pointing to the slot, and the distance to it has to be as short as possible to minimise the remaining errors. It is necessary to pay so much attention to this case because it is the one with the least safety margin in the entire setup.

OpenFridge()

This method moves the mobile platform to the fridge, does the usual classification and fine-positioning over the marker, moves the arm to the handle of the sliding lid, opens it by pushing the handle sideways and then moves the arm back into its park position. Figure 7.10 illustrates this sequence. The opening is divided into two parts:

- During the first part the force sensor is used to check whether the lid gets stuck. A violation of the force limit is treated as an error and the sequence aborted (but this has so far never happened except when deliberately blocking the lid for testing purposes). At the same time the force with which the gripper presses onto the lid is controlled so that it does not loose contact (the lid is slightly tilted in the arm coordinate system). This motion lasts for all but the last 2 cm of the distance. A sample force log file can be seen in figure 7.11(a).

- During the second part the lid has to be pushed over a slight resistance. This requires a lot more force than just sliding it and therefore a different motion which uses a much higher force limit than the first. A violation of this limit is not treated as an error, but only as a reason to terminate the motion because the lid has arrived at its goal. A sample force log file can be seen in figure 7.11(b).

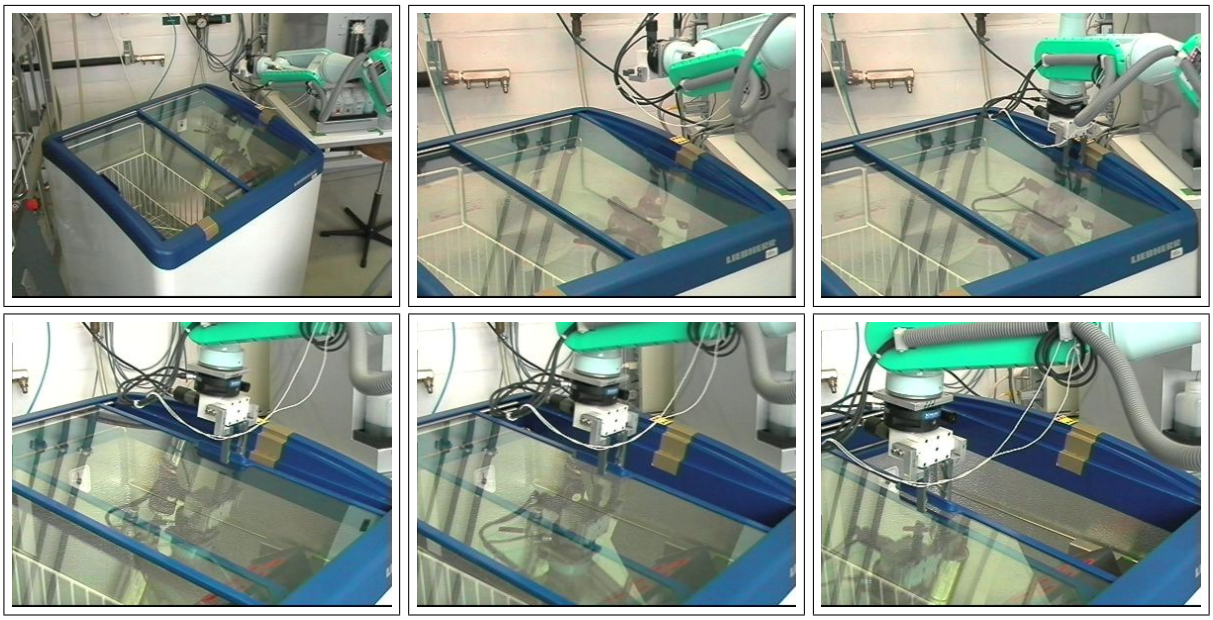
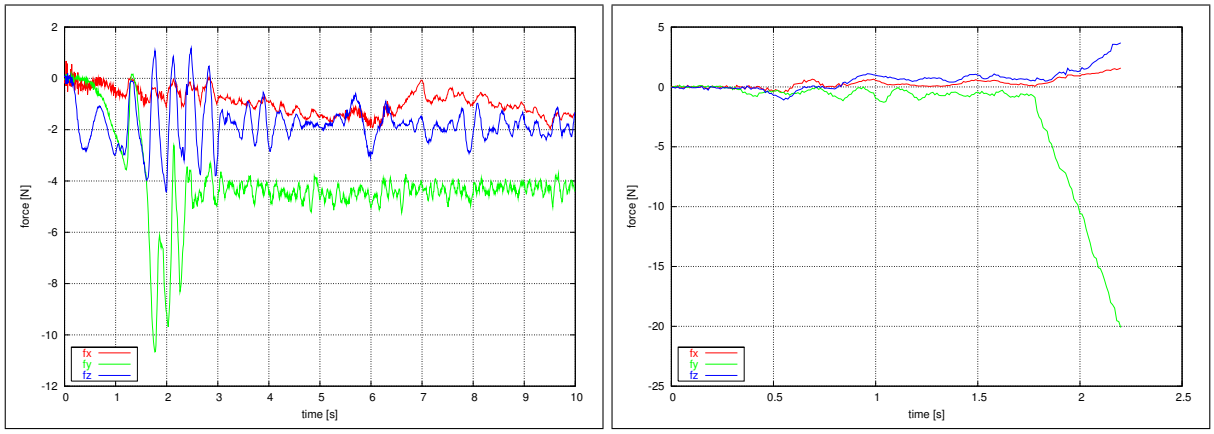


Figure 7.10: OpenFridge()



(a) sliding phase (b) stopping phase

Figure 7.11: OpenFridge() – Force during opening of the sliding lid.

StopAndUnloadCentrifuge()

The second half of centrifuge operations has been integrated in the `StopAndUnloadCentrifuge()` method. It moves the mobile platform to the centrifuge, does the usual classification and fine-positioning, presses the "LID" button to unlock the lid, tries to lift it to see if it is really unlocked (and aborts if it is not, but this has so far never happened), then opens the lid using a circular motion, centres over the cage inside the centrifuge, detects the position of the tube in it, optionally rotates the cage so that the tube is in a proper position[5], grasps the tube from the slot and retreats the arm to the park position. Figure 7.12 visualises these steps.

PlaceTubeWaste()

This method is equivalent to `PlaceCedexWaste()`.

PickTubeStorageBarcode()

This method is equivalent to `PickTubeStorage()`, except that the storage rack for NUNC tubes *with* barcode is used. Apart from their markers these two storage racks are absolutely identical and therefore the methods use identical scripts.

HoldTubeScanner()

This method moves the mobile platform to the barcode scanner, does the usual tasks of classification and fine-positioning and holds the tube in front of the scanner to read its barcode. The arm is left holding the tube after this method returns. It may only be followed by either `RotateTubeScanner()` or `TakeTubeScanner()`.

[5]The optional rotation of the cage is the same as with `LoadAndRunCentrifuge()`, only that now the robot has no tube grasped and therefore does not need to temporarily deposit it anywhere.

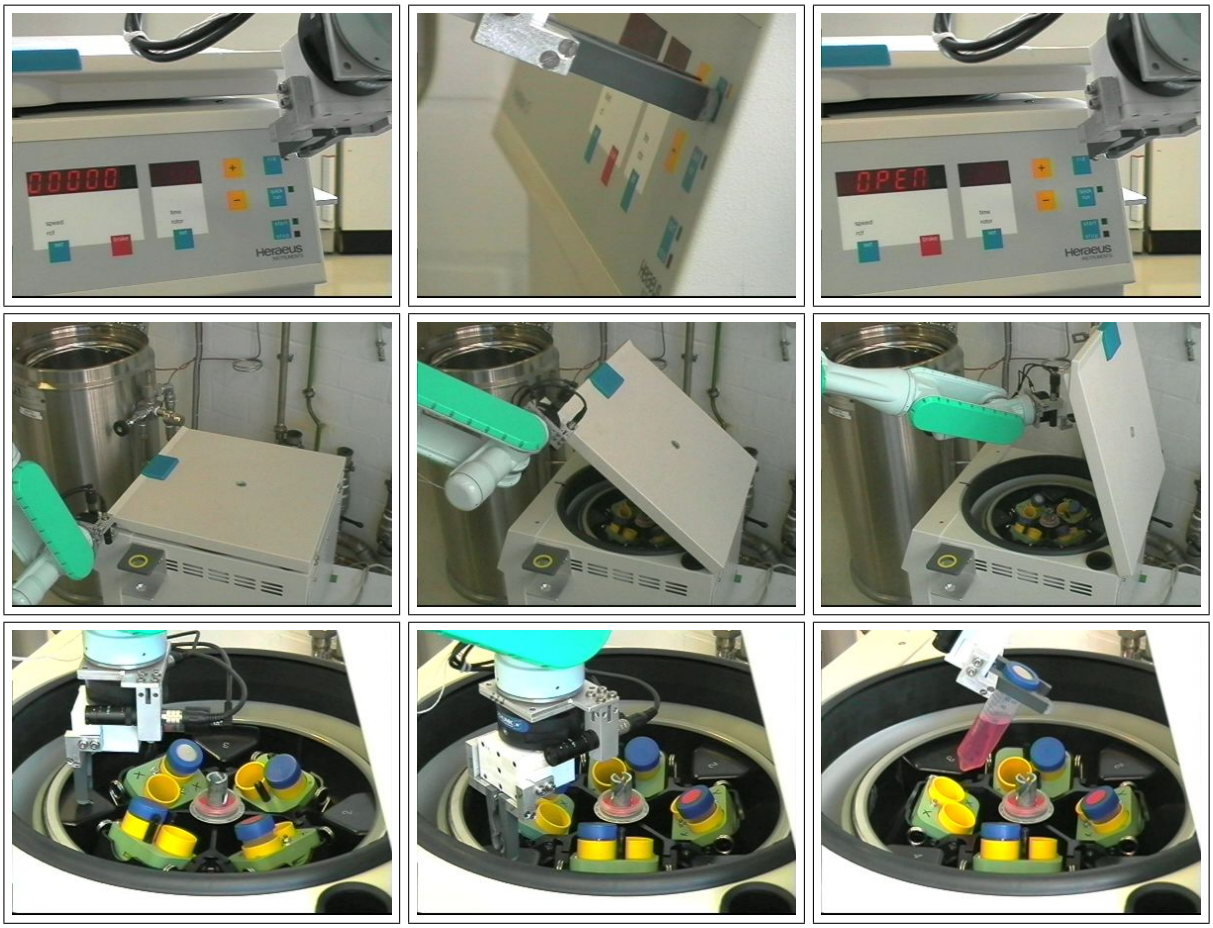


Figure 7.12: StopAndUnloadCentrifuge()

RotateTubeScanner()

In case the barcode scanner should be unable to read the barcode due to damage, dirt or an improper placement of the code this method is used to rotate the tube about the world z-axis by a small random angle. The arm is left holding the tube after this method returns. It can be called several times until the scanner succeeds.

TakeTubeScanner()

This methods takes the tube from the barcode scanner again and moves the arm back into its park position. Since all relevant positions have been stored no vision or force sensor is needed during this operation.

PlaceTubeAndCloseFridge()

The last biotechnological step in the cycle is to archive the supernatant in the fridge by using this method. It moves the mobile platform to the (open) fridge, does the usual classification and fine-positioning, selects a free slot from the rack inside the fridge, places the tube into that slot, moves the gripper behind the handle of the sliding lid, closes it by pushing it sideways and moves the arm back into its park position. Figure 7.13 illustrates these steps.

During insertion of the tube in the slot the force sensor is used in a force controlled peg-in-hole operation to ensure that the tube does not get stuck. It is also used during closing the lid, which happens in the same two steps as with `OpenFridge()`.

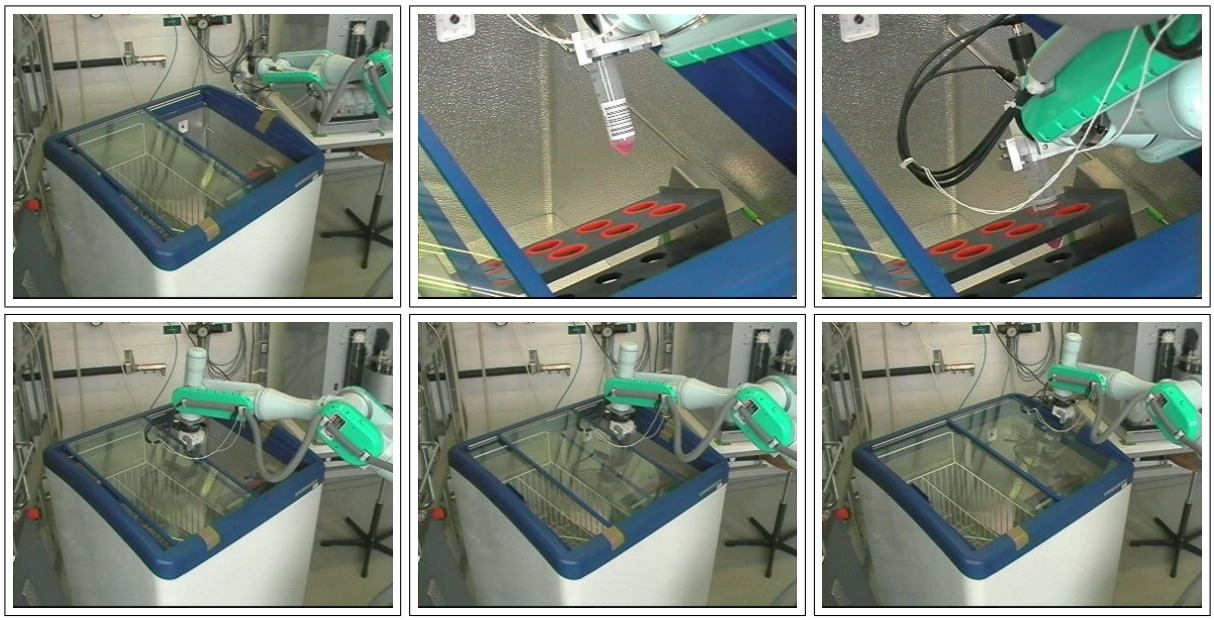


Figure 7.13: PlaceTubeAndCloseFridge()

ParkCharger()

Finally, this method moves the mobile platform to its charging station and docks to it. Similar to `UnparkCharger()`, the collision avoidance is switched off during approaching the station with a relative forward motion. This method may only be followed by `UnparkCharger()`.

7.2 Biotechnological Results

The above presented cycle of robot actions has been tested many times and has been used in test cultivations without major problems. The mobile robot itself has therefore proven its fitness for the desired task, but the automated sample management system affects more than just the robot. The auxiliary devices and the biotechnological evaluation of the cultivations in the project have been the work of I. Poggendorf. To show that the automated sample management also has no impact on the biotechnological parameters this section briefly presents results from her work[6].

Overall, three 20 l cultivations were done with the automated sample management system, one with hybridoma cells and two with *recombinant chinese hamster ovary* (rCHO) cells. Since the cultivations were also used to optimise the system they did not run completely without restarting the control program(s). The maximum number of autonomous cycles without interruption was reached in the second cultivation with seven cycles, corresponding to more than two complete days. During the third cultivation a hardware failure occurred on one of the platform motors after 44.5 h so that only six autonomous cycles could be done.

[6] See [Poggendorf 2004].

During these cultivations three samples were taken per day: One in the early morning, one at noon and one in the late afternoon. In addition to this, during the hybridoma cultivation one sample was taken during the night. The samples taken by the robot system were compared with manually taken samples, which have been counted twice using the Cedex. The main biotechnological difference between the automatic and the manual samples are:

- The samples are taken through a different valve at the reactor, which may affect the cell density.
- The sampling device uses rather long tubes, which may affect the cell density due to dilution with condensate and sedimentation of cells.
- The pipetting uses different mechanisms, which may affect the cell density due to sedimentation of cells and the cell viability due to physical force exerted on the cells.
- The robot needs more time for cycle, which may affect the cell density and viability.

The results from the cell count and viability analysis for the three cultivations are given in figure 7.14. The general decrease of the cell densities between 72 and 96 h in the first, at 48 h in the second cultivation and at 69 h in the third cultivation are dilution effects – bioreactors are not run with their full volume right from the start, but are stepwise filled up because cells do not grow well if their density is very low. In the first cultivation the nutritients in the broth are used up after 144 h and the cells start dying. In the second cultivation a microbiotic infection occurred at at 67.5 h and had to be treated with antibiotics, but the growth rate remained rather low, so that the cultivation was aborted prematurely.

Looking at the cell density the first cultivation has a larger deviation between automatic and manual counts than the other two. This is because the first cultivation used the original version of the pipetting device, whereas the other two used a modified version. In this modification a reservoir holding 1 ml of the sample was added to the tubing system directly above the needle. Without this reservoir cells could sediment in the tubes and the syringe, so that a diluted sample was ejected out of the pipette and fed to the Cedex. With this reservoir the dilution is almost completely avoided and the remaining differences are within acceptable statistical bounds.

A comparison of the viability rates shows that no cell damage occurs with the automatic sample management. The automatic sample management therefore can be considered as equivalent to the manual sample management as far as the main biotechnological parameters are concerned. The sample is not significantly diluted by the automatic system and the cells are not damaged. The still slightly lower cell density with automatic sampling compared to manual sampling should be examined further. It may be caused by statistical effects, in which case more test cultivations will provide the necessary reliability.

7.3 Summary

Several tests have been done with the robot system. The mobile robot has been tested in parts, and the entire robot system during numerous dry runs and three real test cultivations. The tests of the mobile platform revealed that it is well capable of reaching the required positioning accuracy of less than 1 centimetre.

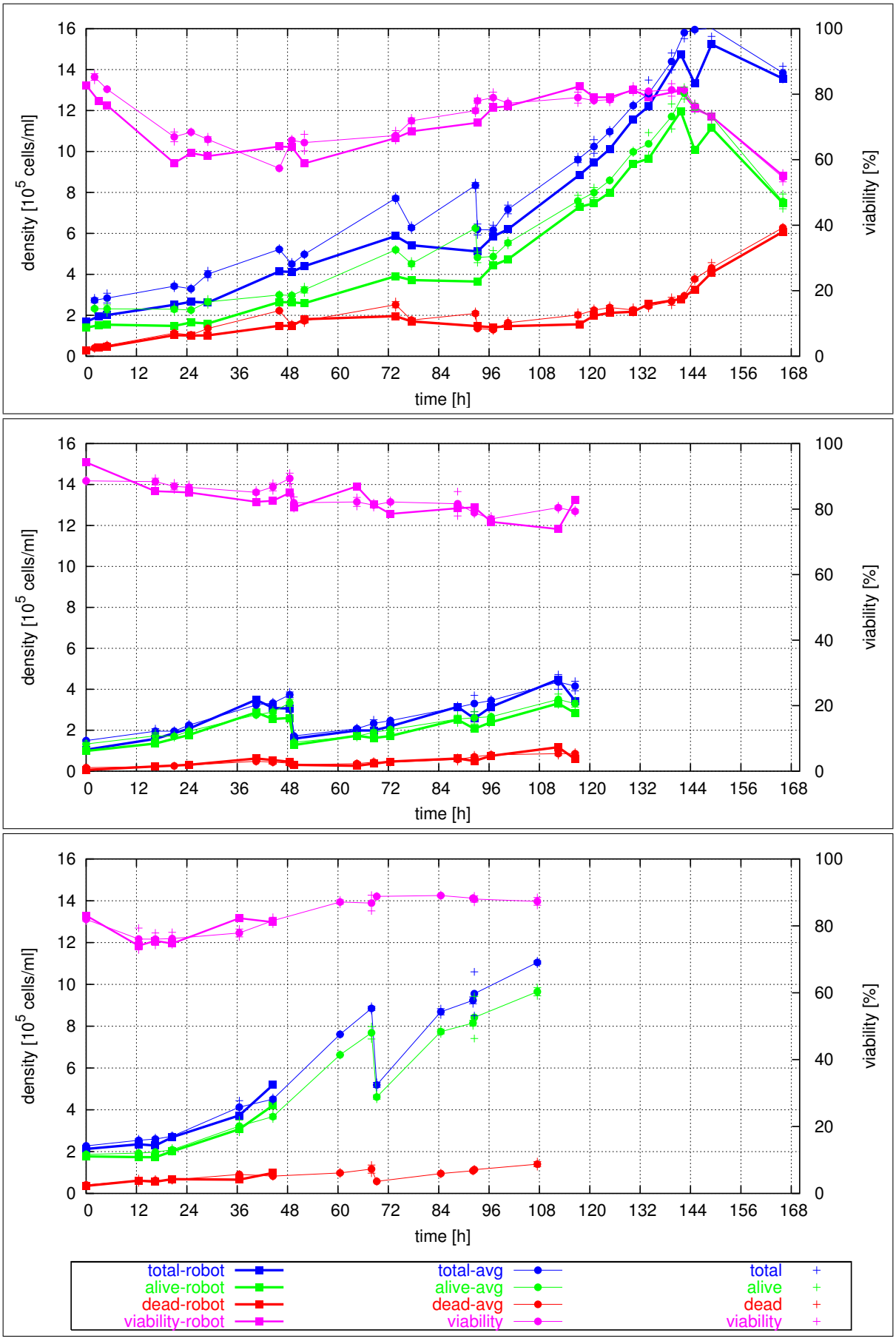


Figure 7.14: Cell density and viability results from the three test cultivations plotted in the same scale from top to bottom: Hybridoma cell cultivation, first rCHO cultivation and second rCHO cultivation.

The vision system of course still needs sufficient, but not standardised illumination. It has so far worked reliably in all practical tests, but still has the potential of improvements, in particular its run-time complexity.

The arm control does not offer much room for improvements because of limitations of the hardware. The PA-10 arm positioning is quite accurate, but compared to older PUMA robots rather sluggish. Unfortunately this means that force control is possible, but remains suboptimal.

Nevertheless, the mobile robot successfully completed three test cultivations – not taking the one mechanical failure into account. During these cultivations no significant differences between automatically and manually taken samples could be found.

According to all these results it can therefore be stated that the implemented mobile robot system is a feasible approach to automating the sample management during biotechnological cell cultivations.

Chapter 8

Conclusions & Future Work

> *"Science has achieved some wonderful things of course, but I'd far rather be happy than right any day."*
>
> *Slartibartfast*

The project goal was to do a feasibility study for establishing a robot system to automate the sample management in a biotechnological laboratory – a complex procedure involving numerous spatially separated devices. As is natural with a feasibility study, there remain a few issues that could impact the practical use of such a robot system and therefore need addressing. Since most of these issues are tightly bound to a specific piece of hardware for which no alternatives existed within the scope of this work they could not be properly covered. These issues are discussed in this chapter.

Since it has proven unfeasible to aggregate the required devices around a stationary robot arm, the arm had to be mounted to a mobile platform to achieve the necessary autonomy. Both the mobile platform and the robot arm have several sensors to detect both their position and the position of the devices the system is supposed to manipulate. The robot should be easy to use and easy to adapt to different sequences because the sample management process may required different steps for different cultivations. All this, together with the fact that the robot is required to co-exist in an environment with human personnel that may accidentally obstruct the robot's way or may accidentally move the devices, makes up for a very complicated system.

Yet, it has been established, and so the approach has to be considered feasible and the project a success.

The system has passed the component tests as well as several test cultivations as presented in chapter 7. The conclusions are:

- It *is* possible to use a standard robot arm to manipulate tubes and devices that have not been designed to be manipulated by a robot.
- It *is* possible to design a vision system that can detect objects that have not been designed to be detected by a camera without standardised illumination.
- It *is* possible to position a mobile platform with an accuracy of less then 1 centimetre that has not been designed to be positioned that accurately.
- And, finally, it *is* possible to integrate a robot system into a greater distributed system in a way that requires only limited knowledge about details.

8.1 The Robot System

In order to establish an entire robot system in the limited time and with the limited human resources available in the project several constraints had to be applied. The most important constraint was the explicit intention to buy as much equipment as possible ready-to-use and "only" deal with integrating it into a robot system and the high-level issues. This, it turned out, does not nearly work as easily as expected. The problem is that commercial hardware is usually designed for very specific tasks, and it is something between very hard and impossible to see by looking at the official glossy sales brochures (specifications) whether they are suited for something else and more experimental too. Often the specs suggested functionalities that later on could not be achieved. On the other hand side the limited funds of the project did not allow to upgrade the hardware once these deficiencies were recognised, and in fact it often turned out that better hardware was not available at all. This makes up for sometimes unsatisfying solutions that do affect the overall system performance. It certainly did increase the amount of software that had to be designed and implemented to way beyond what was intended.

About the Arm

For example, the PA-10 robot arm was and still is the one best suited to be mounted on a mobile platform because of the small size of its controller and its power supply. The software manual also makes it look suitable by mentioning the *each axis real time control mode* – for each axis a new desired position value can be given in each cycle. This means that everyone who does not want to use Mitsubishi's software natively thinks that it is going to be possible to simply add support for this mode to their own software and not have to bother with the low-level details. As is demonstrated by the results in sections 4.4 this is not true because the resulting performance is too poor for virtually any application.

In this work this has led to the requirement to add full PA-10 support to RCCL, which has taken a considerable and unscheduled amount of time. It also meant the introduction of the PID joint controller, which has so far been the unavoidable source of inaccuracies. The performance of the implemented solution is several times better than with Mitsubishi's software, but still suboptimal. This particularly manifests itself in the force control that is used in (only) some parts of the sequences: With the sluggish reaction of the arm only very slow force control is possible.

One might think that robot arms are so well-established products that these problems do not occur, but this is obviously not the case. Yet, no real alternatives exist because other robot arms do not nearly as easily fit on a mobile platform. The conclusion therefore has to be that the PA-10 still is the best-suited arm for these applications, but that more effort has to be spend on optimising the control software used to access this arm.

About the Platform

For the mobile platform similar considerations apply. The initial idea of buying the platform complete with software could not be pursued because the accuracy of the supplied software did not meet the requirements. The vendor did improve the software during the course of this work, but the schedule did not allow to wait for this. As a result an in-house solution had to be developed, which again took a considerable and unscheduled amount of time.

The extended kalman filter (EKF) used in this solution is a somewhat conservative approach to the localisation problem, compared to techniques like particle filters, but fully sufficient. It does, however, share two problems with them: First, since the model contains an integrator (from velocity to position) the filter can not be guaranteed to be stable in the sense of "bounded input" → "bounded output". Second, since the laser measurements are only *relative* measurements (distances and angles) an initial guess about the position must exist in order to associate a laser scan with a map.

The latter is a generic measurement problem and not specific for any particular filter type. When humans navigate by landmarks they usually recognise them individually. That means, they recognise buildings, churches, trees, rivers, mountains or whatever because they have seen them before or can identify them on a map. With laser reflector markers the mobile platform does not recognise *individual* landmarks, but only that there *are* landmarks at all. It does not know which landmark in the scan is which landmark in the map because they all look the same – they do not carry any information that allows them to be distinguished. The only way to associate them is to have an initial guess about the position and then use the measurements to only correct that guess. If this initial guess does not exist or is too poor the association can not be established and the filter will fail. Only a means to measure an *absolute* position or to make the landmarks unique can avoid this problem, but none of this is possible with the given hardware.

What remains is that the current mobile platform control software has passed all tests, but can not be guaranteed to always do so. The conclusion therefore has to be that in order to gain any kind of guarantee additional absolute sensors are needed.

About the Vision

The vision system is the component of the system that least suffers from its hardware. According to the tests in sections 5.5 its accuracy is better than the positioning accuracy of the robot arm. By focusing on the centre of gravity (COG) of coloured regions – which can be extracted very robustly – it achieves a very high reliability without requiring standardised illumination. Contrary to many other systems the usual laboratory illumination is therefore fully sufficient. If the vision system does fail because of insufficient illumination this can be easily detected and the operation can be repeated. Problems like with the mobile platform where one step can lead to a faulty situation which the next step can not cure are therefore almost impossible.

On the other hand the image quality of the given hardware is quite poor compared with modern digital still cameras, especially the colour saturation and noise. The camera used is a microhead (lipstick) camera with a single CCD sensor. There are microhead cameras with 3 CCD sensors to grab RGB colours separately, but these cameras are very expensive. Other cameras with 3 CCD sensors exist too, but are of partly substantially larger size. Since the arm currently uses almost all its operational range to manipulate the devices a larger tool that requires a larger safety distance is a potentially fatal idea.

Another aspect is the image data transmission. The microhead camera uses analog video signal transmission and requires the images to be digitised with a framegrabber, which does not improve the image quality. A digital transmission like IEEE 1394 (firewire) would improve the image quality significantly, but the currently available firewire cameras are also of larger size. Advances in camera technology can be expected to improve this situation.

Since the coloured markers used for the region detection suffer from the same association problem as the localisation of the mobile platform an approach has been implemented that tries all possible permutations of associations and choose the best solution. This approach is very stable, but also numerically quite expensive. If the image is distorted by too many background or erroneous regions the detection can take several seconds up to minutes – an upper bound is not given[1].

The conclusion therefore has to be that using a different camera is definitely a worthy idea to improve the image quality and thus the reliability, but that an increased size of the camera may lead to problems. Compared to the drawbacks of the other components this one is currently of minor importance. What is of importance is the execution speed of the vision software, and here an optimisation of the mapping process is strongly recommended. Amongst the possible improvements are partial mappings to reduce the computation speed and the idea to use the expected position as initial guess to reduce the complexity.

Speed Issues

Speed is also an issue with the robot motions as such. Currently, one automated sample management cycle takes up to 45 minutes, which is about twice the time a human operator would need. For a single reactor cultivation of mammalian cells under pilot scale constraints this is not a problem because these cells grow rather slowly, but for other cells or more than just one reactor it may be too long. Ways to speed up the sequence are therefore an interesting and important topic.

The mobile platform runs at a translational speed of 0.5 m/s, which is almost its technical limit. As far as the mobile platform is concerned a speedup can only be achieved by avoiding motions wherever possible. The setup used in the test cultivations already took advantage from placing devices together on different tables at what appears to be the same position for the mobile platform, but the layout can still be optimised. On the other hand, perhaps only 1 or 2 minutes could be gained by this, so the arm motions must also be optimised.

The nominal arm speed (30 deg/s) should not be increased because of the issues with the PID joint controller, especially not the force controlled motions. The only way to speed up arm motions is therefore also to avoid them. The setup used in the test cultivations already incorporated this optimisation for single devices. The most prominent examples are the `LoadAndRunCentrifuge` and `StopAndUnloadCentrifuge` commands, which group together a set of atomic operations. Since each of the atomic operations goes into a park position after completition only to leave that position and recentre over the centrifuge immediately again for the next atomic operation a lot of time can be saved by optimising the sequence and dropping all redundant park/unpark and centring motions.

This idea could be extended to also optimise motions involving several devices. Currently the arm goes back into a park position after each command, which – considering the above combination of operations – means after it is done with a device. Since many devices are already grouped spatially together to avoid platform motions the arm could take advantage of this and also drop redundant park/unpark motions between devices that are at the same platform position. Depending on how optimal the devices are grouped together several minutes may be gained by this.

[1] This is also the reason why this approach is not applicable to the localisation.

Combined, the execution speed can perhaps be brought down to the time a human operator needs for the task, but obviously a tradeoff remains: The more optimised the setup and the robot motions are the more specific they are to the given situation. With such an optimised system a slight modification of the sequence will result in much larger changes of the scripts than when atomic operations are used. This is not a fundamental problem because both sequences can exist at the same time, but it is up to the user to decide which one to prefer.

The only really larger speedup that can be achieved while still using atomic operations is with parallelisation. Currently the robot has to wait at more than just one point in the sequence for other devices to become ready. With just one reactor there is only limited room for parallelisation, but with more reactors it will for example be possible for the robot to fetch a sample from the second reactor while the pipette is still cleaning itself from the sample from the first one. This could bring down the average cycle time significantly and make the system more efficient.

Operating System Issues

Another issue is the operating system (OS) that is used on the computer controlling the arm and the platform. Currently a plain Linux 2.4 kernel is used, which is officially not real-time capable. Early tests with rather old hardware even before the beginning of this work showed that the Linux kernel can have a scheduling latency of up to several milliseconds, with no upper bound. Clearly this would be unacceptable for any robot control, but with newer hardware and newer kernels this situation has improved a lot. Contemporary hard/software combinations yield a typical latency of less than 1 millisecond, though still without a guarantee. Linux kernels 2.6 and above are preemtible, which is an important step towards even further decreasing the latency, though even still without a guarantee. If hard real-time requirements should ever become an issue, an alternative OS like QNX or Solaris must be evaluated.

For this work these alternatives are not used because with Linux the driver support for additional hardware is on average much better, and if it is not, the existence of free source code helps in implementing new or modifying existing drivers. In addition to this hard real-time requirements are practically speaking not strictly required by the system. Due to the PID joint controller the PA-10 arm is sufficiently sluggish so that it goes completely unnoticed if a setpoint should be missed by up to several milliseconds, and the mobile platform is even more sluggish. As a result a plain 2.4 Linux kernel has so far proven itself sufficient.

Biotechnological Issues

Apart from laboratories at universities like the one used during the test cultivations biotechnological production involves clean rooms, which is an issue with the current robot. Clean room technology is about minimising particles in the air, to which germs might attach and contaminate a cultivation. For any equipment brought into or installed in a clean room this means that a certificate must exist that it does not emit more than a certain number of particles of a certain size. In addition to this it must be sterilisable by either ethanol (or any other liquid detergent) or hot pressurised stream. Both of this is currently not possible with the robot system because of several fans ensuring its cooling and the fact that its electronic and mechanic components are not sealed against liquids. To allow operation of the robot system in a clean room major design modification therefore have to be done.

8.2 Roblets

One interesting offspring inspired by the experience with remote programming of mobile robots gained during this project is the ROBLET® architecture by H. Stanek[2], an alternative for the script programming. As has been mentioned in section 6.1 the robot command scripts have been implemented to provide the highest possible level of flexibility at the lowest possible level of complexity. Such an approach necessarily leads to a somewhat limited level of functionality. The scripts have never been intended to be seen as a real programming language, but a real programming language is what is ultimately required to provide an ultimately satisfying level of functionality. Such a programming language has been regarded as out of the scope of this work because its details would distract the non-expert user from its real task – the "programming" of the automated sample management.

The ROBLET® architecture addresses exactly that problem by using the Java™ language and encapsulating all robot details in *units*[3]. Java™ is a widely known and used object-oriented language invented by Sun Microsystems[4]. It has especially been designed with distributed systems in mind and therefore includes a lot of useful encapsulations of network details. Java™ programs are compiled into an intermediate language called *byte code*, which is executed by the Java™ Virtual Machine (JVM). This JVM exists for a large variety of computers and operating systems, including embedded systems.

Since Java™ is so commonly known it presents a very good compromise between functionality and the need to learn a programming language. Chances are that users programming a robot system already have knowledge about Java™. If they do not, the effort to learn it is not wasted on a very specific language that they are unlikely to ever use again for anything else, but is spend on a very versatile language that may be useful for many other future occasions. Because of this the "distraction" argument concerning the amount of effort that has to be spent on learning a new programming language can be considered inapplicable to Java™. Anyway, only a very limited amount of Java™ knowledge is needed because ROBLET®s encapsulate most low-level details.

The ROBLET® architecture consists of a ROBLET® server on the mobile robot and one (or more) clients on the network as in figure 8.1. Instead of requesting functionality from the server by using *remote procedure calls* (RPCs) the clients make use of an interesting feature of Java™ – the possibility to send real application *code* to the server. Once received, the ROBLET® server executes these ROBLET®s in its own JVM. This execution of code makes this approach fundamentally different from all other networking tools like CORBA, which only send *data* describing the requests.

The ROBLET® server provides the ROBLET® with a set of *units* that manifest the "world" as the ROBLET® sees it. These units represent all sensor, actor and helper methods that may be used by the ROBLET® to implement its functionality[5]. The units are expected to represent only *abstract* interfaces to the hardware in order to keep the ROBLET®s free from low-level details. By first requesting which units are present the ROBLET® can thus be kept independent from any specific assumptions and can run on a wide range of robots.

[2]Das genRob-Projekt, Hagen Stanek, Albrecht-Dürer-Str. 16, D-71065 Sindelfingen, Deutschland. `http://www.genrob.com`

[3]See [Westhoff et. al. 2004a] and [Westhoff et. al. 2004b].

[4]See `http://java.sun.com`.

[5]Java™ class loaders can be used to ensure that the ROBLET® is caged in a kind of secondary *sandbox* and can not call other methods except those from the units.

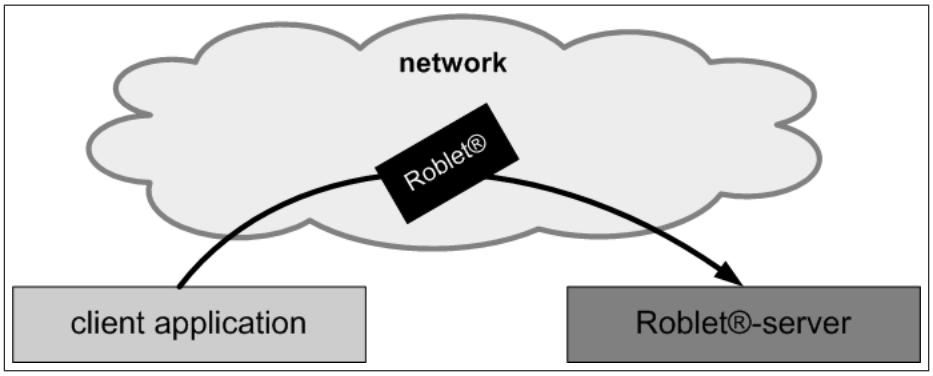


Figure 8.1: The ROBLET® architecture.

Currently, the ROBLET® architecture is in an experimental phase of implementation and supports access to the mobile platform only. In the future units encapsulating RCCL on the PA-10 arm, the vision system and the INFOBASE interface will be added. It will then be possible to access the entire functionality of the robot system using ROBLET®s, but the goal is to keep both interfaces available in parallel. The major advantage of the ROBLET® approach will be that users can use the Java™ language to implement much more refined computations of transforms, safety checks and conditional branching than are possible with the script approach. This will significantly simplify usage of the robot system and is even likely to open completely new areas of application.

8.3 Applications

One question that has often arisen in the presentations of the robot system to an interested audience is whether it can not be applied to other types of laboratories as well. The answer is of course “yes”, but with a “but”.

First of all, the robot does not see what it is doing. This means that as long as the provided primitives allow to manipulate different devices the modifications of the scripts are straightforward. Whether this covers additional biotechnological devices or, for example, devices used in a chemical laboratory does not matter at all. The “problems” only start to mount when the implemented primitives do not suffice and new ones have to be added, and even then it does not matter which type of device they are meant to be used for. Basically, the simpler the operations are the less the effort that has to be spend on adapting the robot system is. Apart from that, anything is possible – at least unless it requires visual services that go beyond the currently implemented ones.

The question whether the robot system can be used for different tasks is therefore reduced to the question whether these tasks require similar manipulations, where “similar” means similar in a low-level meaning. Two tasks that require placing a tube into some device are not necessarily similar, because they may require completely different motions. In general, other devices can be manipulated and other laboratories can be automated, but a detailed statement about the required modifications can only be made for each new setup individually.

Concerning the practical applicability of the mobile robot system and its commercial potential, Bayer Healthcare in Berkeley (USA) is currently planning a new production facility designed to be suitable for a mobile robot. Encouraged by the results from the system presented

in this work they are considering to evaluate this new automisation approach for some of their processes. This will be the first time ever that a mobile robot system enters a biotechnological production plant – a step that will produce valuable results for the further development of the robot system and might change the automation forever.

Bibliography

[Adams 1979] **D. N. Adams (1979)**
"The Hitch-Hikers Guide to the Galaxy"
Pan Books Ltd, London, UK.
ISBN 0-330-25864-8

[Arkin 1998]
R. C. Arkin (1998)
"Behaviour Based Robotics"
MIT Press, ISBN 0262011654.

[Bar-Shalom and Li 1993]
Y. Bar-Shalom and X.-R. Li (1993)
"Estimation and Tracking: Principles, Techniques and Software"
Artech House Inc., Norwood, MA 02062, USA
ISBN 0-89006-643-4

[Berger 2000]
C. Berger (2000)
"Ein robuster, farbbasierter Objekterkenner zur Roboterfeinpositionierung in einem realen Laborumfeld"
Diplomarbeit, Universität Bielefeld, Technische Fakultät, Germany.

[Borenstein, Everett and Feng 1996]
J. Borenstein, H. R. Everett and L. Feng (1996)
"Where am I? – Sensors and Methods for Mobile Robot Positioning"
`http://www-personal.engin.umich.edu/~johannb`
`http://www-personal.engin.umich.edu/~johannb/position.htm`
`http://www-personal.umich.edu/~johannb/shared/pos96rep.pdf`

[Borenstein and Feng 1995]
J. Borenstein and L. Feng (1995)
"UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots"
Proceedings of the 1995 SPIE Conference on Mobile Robots, Philadelphia, October 22-26, 1995
`http://www-personal.engin.umich.edu/~johannb`
`http://www-personal.engin.umich.edu/~johannb/umbmark.htm`
`http://www-personal.engin.umich.edu/~johannb/Papers/paper60.pdf`

[Borenstein and Feng 1996]
J. Borenstein and L. Feng (1996)
"Gyrodometry: A New Method for Combining Data from Gyros and Odometry in Mobile Robots"
Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, Minnesota, Apr. 22-28, 1996, pp. 423-428.
`http://www-personal.engin.umich.edu/~johannb`
`http://www-personal.engin.umich.edu/~johannb/gyrodom.htm`

[Collani 2001]
Y. O. v. Collani (2001)
"Repräsentation und Generalisierung von diskreten Ereignisabläufen in Abhängigkeit von Multisensormustern"
Dissertation, Technische Fakultät, Universität Bielefeld, Deutschland.
`http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:361-1657`

[Denavit and Hartenberg 1955]
J. Denavit and R. S. Hartenberg
"A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices
Journal of Applied Mechanics, June 1955, pp. 215 - 221.

[Denavit and Hartenberg 1964]
J. Denavit and R. S. Hartenberg 1964
"Kinematic Synthesis of Linkages"
McGraw-Hill, New York, 1964.

[Doucet, Freitas and Gordon 2001]
A. Doucet, N. de Freitas and N. Gordon (2001)
"Sequential Monte Carlo Methods in Practice"
Springer.

[Ferch 2002]
M. C. Ferch
"Lernen von Montagestrategien in einer verteilten Multiroboterumgebung"
Dissertation, Technische Fakultät, Universität Bielefeld, Deutschland.
`http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:361-3722`

[Fu, Gonzales and Lee 1987]
K. S. Fu, R. C. Gonzales and C. S. G. Lee (1987)
"Robotics – Control, Sensing, Vision and Intelligence",
ISBN 0-07-022625-3, McGraw-Hill International Editions, Singapore.

[Gallistel 1990]
C. Gallistel (1990)
"The Organisation of Learning"
The M.I.T. Press, Cambridge, Massachusetts and London

[Graf 2000]
T. Graf (2000)
"Flexible object recognition based on invariant theory and agent technology"
Dissertation, Technische Fakultät, Universität Bielefeld, Deutschland.
`http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:361-1408`

[Hart, Nilsson and Raphael 1968]
P. Hart, N. Nilsson and B. Raphael (1968)
"A formal basis for the heuristic determination of minimum cost paths"
IEEE Transactions on Systems, Science and Cybernetics, 4(2), p.100-107.

[Hunn 1993]
U. Hunn (1993)
"Implementierung von Planung und Steuerung der Bewegung eines autonomen Fahrzeuges in einer Fuzzy-Umgebung"
Diplomarbeit, Institut für Prozessrechentechnik und Robotik, Universität Karlsruhe.

[IEEE 1985]
IEEE Computer Society (1985),
"IEEE Standard for Binary Floating-Point Arithmetic",
IEEE Std 754-1985.
`http://standards.ieee.org`

[JAI CV-M1000 Series Operation Manual]
JAI A.S.
CV-M1000 Series Operation Manual (2700-21092 E A2 0399)
JAI A.S., Produktionsvej 1, DK-2600 Glostrup, Copenhagen, Denmark.

[Julier and Uhlmann 1997]
S. Julier and J. K. Uhlmann (1997)
"A new extension of the Kalman filter to nonlinear systems"
Proceedings of AeroSense, 11th International Symposium on Aerospace/Defense Sensing, Simulations and Controls.

[Kalman 1960]
R. E. Kalman (1960)
"A new approach to linear filtering and prediction problems"
Transactions of the ASME, Journal of Basic Engineering 82, p. 35-45

[Khatib et.al. 1996]
O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg, A. Casal and A. Baader (1996)
"Force strategies for cooperative tasks in multiple mobile manipulation systems"
In G. Giralt and G. Hirzinger (editors)
Robotics Research: The Seventh International Symposium, p. 333-342.

[King et. al. (2004)]
R. D. King, K. R. Whelan, F. M. Jones, P. G. K. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell and S. G. Oliver
"Functional genomic hypothesis generation and experimentation by a robot scientist"
Nature, Vol. 427, p. 247, 15. January 2004.

[Lee 1996]
D. Lee (1996)
"The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Robot"
Cambridge University Press, Cambridge

[Lloyd and Hayward 1989]
J. E. Lloyd and V. Hayward (1989)
Multi-RCCL User's Guide
McGill Research Centre for Intelligent Machines, McGill University, Montréal, Quebéc, Canada.
`http://www.cs.ubc.ca/spider/lloyd/rccl`

[Lloyd 1995]
J. E. Lloyd (1995)
"Robot Trajectory Generation for Paths with Kinematic Singularities"
Ph. D. dissertation, Department of Electrical Engineering, McGill University, Canada.

[Lloyd and Hayward 1996]
J. E. Lloyd and V. Hayward (1998)
"Discrete Algorithm for Fixed-path Trajectory Generation at Kinematic Singularities"
Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis (Minnesota), USA, pp. 2743 - 2748.
`http://www.cs.ubc.ca/spider/lloyd/papers.html`

[Lloyd 1998]
J. E. Lloyd (1998)
"Removing the Singularities of Serial Manipulators by Transforming the Workspace"
Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp. 2935 - 2940.
`http://www.cs.ubc.ca/spider/lloyd/papers.html`

[Lloyd and Hayward 1998]
J. E. Lloyd and V. Hayward (1998)
"Generating Robust Trajectories in the Presence of Ordinary and Linear-Self-Motion Singularities"
Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp. 3228 - 3234.
`http://www.cs.ubc.ca/spider/lloyd/papers.html`

[Lloyd 1998]
J. E. Lloyd (1998)
"Trajectory Generation implemented as a Non-linear Filter"
Technical Report 98-11, Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada.
`http://www.cs.ubc.ca/cgi-bin/tr/1998/TR-98-15.pdf`

[Lütkemeyer et. al. 2000a]
D. Lütkemeyer, I. Poggendorf, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2000)
"Robot Automation of Sampling and Sample Management in Pilot Scale"
Cell Culture Engineering VII, Santa Fe, USA, 5. bis 10.3.2000

[Lütkemeyer et. al. 2000b]
D. Lütkemeyer, I. Poggendorf, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2000)
"Robot Automation of Sampling and Sample Management in Pilot Scale"
11th International Biotechnology Symposium, Berlin, Germany.

[Lütkemeyer et. al. 2000c]
D. Lütkemeyer, I. Poggendorf, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2000)
"First Steps in Robot Automation of Sampling and Sample Management during Cultivation of Mammalian Cells in Pilot Scale"
Biotechnol. Prog. 16, No. 5, 822-828, 2000

[Lütkemeyer et. al. 2001]
D. Lütkemeyer, I. Poggendorf, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2001)
"Robot Automation of Sampling and Sample Management during cultivations of Mammalian cells in Pilot Scale"
17th ESACT-Meeting, Tylosand, Sweden.

[Minsky and Papert 1969]
M. L. Minsky and S. A. Papert (1969)
"Perceptrons"
MIT Press, Cambridge, MA (USA).

[Mitsubishi PA10 Operating Manual]
Mitsubishi Heavy Industries Ltd.
Portable General Purpose Intelligent Arm Operating Manual (English Translation, 91-000066, Rev. 0)
Mitsubishi Heavy Industries Ltd.,Kobe Shipyard & Machinery Works, Machinery Sales Department, Machinery & Space Systems Sales Section, 1-1 Wadasaki-cho 1-chome, Hyogo-ku, Kobe 652-0854, Japan.

[Mitsubishi PA10 Servo Driver Manual]
Mitsubishi Heavy Industries, Ltd.
Instruction Manual for Servo Driver, SKC-GC20004 Rev. 0
Mitsubishi Heavy Industries Ltd., Head Office, Laser & Electronics Group, Turbomachinery & General Machinery Department, Machinery Headquarters, 5-1 Marunouchi 2-chome, Chiyodaku, Tokyo 100-8315, Japan.

[Moore 1920]
E. H. Moore (1920)
"On the reciprocal of the general algebraic matrix"
Bull. Amer. Math. Soc., 26:394-395.

[Newman 1999]
P. Newman (1999)
"On the Structure and Solution of the Simultaneous Localisation and Map Building Problem"
Ph.D. Thesis, Australian Centre for Field Robotics, The University of Sydney, 2006 NSW, Australia.
`http://www.acfr.usyd.edu.au/people/postgrads/pnewman`
`http://www.robots.ox.ac.uk/~pnewman`

[Niemann 1983]
H. Niemann (1983)
"Klassifikation von Mustern"
ISBN 3-540-12642-2, Springer Verlag, Berlin, Deutschland.

[Penrose 1955]
R. Penrose (1955)
"A generalized inverse for matrices"
Proceedings of the Cambridge Philosophical Society, 51:406-413.

[Plahl 1998]
S. Plahl (1998)
"Automatisierung im Labor. Planung einer Roboter- Arbeitszelle und Entwicklung einer vollautomatischen sterilisierbaren Probenentnahme"
Diplomarbeit, Technische Fakultät, Universität Bielefeld, Deutschland.

[Poggendorf et. al. 2001]
I. Poggendorf, D. Lütkemeyer, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2001)
"Using an Industrial Robot Arm for Monitoring Cultivations of Mammalian Cells in Pilot Scale"
8th International Conference on Computer Applications in Biotechnology, Québec City, Canada.

[Poggendorf and Scherer 2002]
I. Poggendorf and T. Scherer (2002)
"Roboter arbeiten auch am Wochenende gerne - Serviceroboter für die biotechnologische Produktion von Zellkulturen
BioTec - Fachmagazin für Biotechnologie, Nr. 9-10, 36-37, Vereinigte Fachverlage GmbH, Mainz, Deutschland.

[Poggendorf et. al. 2002a]
I. Poggendorf, D. Lütkemeyer, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2002)
"Let's Start Moving - A Mobile Robot has the Potential of Complete Automation in Monitoring Mammalian Cell Cultures.'
GVC/Dechema Jahrestagungen 2002, Wiesbaden, Germany.

[Poggendorf et. al. 2002b]
I. Poggendorf, D. Lütkemeyer, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2002)
"Let's Start Moving - A Mobile Robot has the Potential of Complete Automation in Monitoring Mammalian Cell Cultures"
Cell Culture Engineering VIII, Snowmass, USA.

[Poggendorf et. al. 2003]
I. Poggendorf, D. Lütkemeyer, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2003)
"Einsatz eines Serviceroboters zur Überwachung von Kultivierungen tierischer Zellen in einer Technikumsumgebung"
6. Dresdner Sensor-Symposium (6. DSS), Dresden, Deutschland.

[Poggendorf et. al. 2004]
I. Poggendorf, D. Lütkemeyer, T. Scherer, J. Zhang, A. Knoll and J. Lehmann (2004)
"Automatic 24/7 Monitoring of Cell Culture Behaviour in Pilot Scale Cultivations of Mammalian Cells Using an Autonomous Service Robot"
Cell Culture Engineering IX, Cancun, Mexico.

[Poggendorf 2004]
I. Poggendorf (2004)
"Einsatz eines Serviceroboters zur Automatisierung der Probenentnahme und des Probenmanagements während Kultivierungen tierischer Zellen in einer Technikumsumgebung"
Doktorarbeit, Technische Fakultät, Universität Bielefeld, Deutschland.

[Poynton 2002]
C. Poynton (2002)
"The Color FAQ"
Frequently Asked Questions about Color.
`http://www.poynton.com/ColorFAQ.hmtl`

[Röben 2003]
F. Röben (2003)
"Farbbasierte Regionensegmentierung zur Feinpositionierung eines Serviceroboters in einer Laborumgebung"
Diplomarbeit, Technische Fakultät, Universität Bielefeld, Germany.

[Salmond, Gordon and Smith (1993)]
D. J. Salmond, N. J. Gordon and A. F. M. Smith (1993)
"Novel approach to nonlinear / non-Gaussian Bayesian state estimation"
Radar and Signal Processing, IEE Proceedings F, vol. 140, no. 2, p. 107-113, ISSN 0956-375X.

[Scheering 2000]
C. Scheering (2000)
"Multi-Agenten in einem situierten künstlichen Kommunikator"
Dissertation, Technische Fakultät, Universität Bielefeld, Deutschland.

[Scherer 1998]
T. Scherer (1998)
"Design and Implementation of a Trajectory Generator for Arbitrarily Moving Targets and On-Line Singularity Robustness"
Diploma Thesis, Technische Fakultät, Universität Bielefeld, Bielefeld, Germany.

[Scherer et. al. 2000]
T. Scherer, J. Zhang, A. Knoll, I. Poggendorf, D. Lütkemeyer and J. Lehmann (2000)
"Roboterisierung des Probenmanagements in der Biotechnologie"
Prozessmesstechnik in der Biotechnologie, Vortrags- und Diskussionstagung, Bamberg 2000

[Scherer et. al. 2003]
T. Scherer, I. Poggendorf, A. Schneider, D. Westhoff, J. Zhang, D. Lütkemeyer, J. Lehmann and A. Knoll (2003)
"A Service Robot for Automating the Sample Management in Biotechnological Cell Cultivations"
Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2003, Vol. 2, p. 383-390), Lisbon, Portugal, 16-19 September 2003. IEEE Catalog Number 03TH8696, ISBN 0-7803-7937-3.

[Schmidt 1970]
S. Schmidt (1970)
"Computational techniques in kalman filtering"
In: Theory and Applications of Kalman Filtering, AGARDograph 139, NATO Advisory Group for Aerospace Research and Development, London

[Schmidt 1976]
S. Schmidt (1976)
"Practical aspects of kalman filtering implementation"
In: AGARD-LS-82, NATO Advisory Group for Aerospace Research and Development, London

[Schneider and Westhoff 2002]
A. Schneider and D. Westhoff (2002),
"Autonomous Navigation and Control of a Mobile Robot in a Cell Culture Laboratory",
Diploma Thesis, Faculty of Technology, University of Bielefeld, Germany

[Sedgewick 2001]
R. Sedgewick (2001)
"Algorithms in C" Addison-Wesley, ISBN 0201316633.

[SICK AG 2000]
Sick AG (2000)
"Definition der Telegramme zwischen Benutzerschnittstelle und LMS- oder LMI 400-System über RS422/RS232, Version 02.04.2000"
Sick AG, Auto Ident Division, Nimburger Str. 11, D-79276 Reute, Germany
`http://www.sick.de/de/products/categories/auto/en.html`

[SICK AG 2002]
Sick AG (2002)
"Lasermesssysteme LMS 200/211/220/221/291, Technische Beschreibung"
Item #8008969 (de) / #8008970 (en), Sick AG, Auto Ident Division, Nimburger Str. 11, D-79276 Reute, Germany
`http://www.sick.de/de/products/categories/auto/en.html`

[Smith and Brady 1995]
S. M. Smith and J. M. Brady (1995)
"SUSAN - A new approach to low level image processing"
Technical Report TR95SMS1c, University of Oxford.

[Sobel 1990]
I. Sobel (1990)
"An isotropic 3×3 image gradient operator"
In H. Freeman (editor): "Machine Vision for Three-Dimensional Scenes", pages 376-379, Academic Press, 1990.

[Wan and Merve 2001]
E. A. Wan and R. van der Merve (2001)
"The Unscented Kalman Filter"
to appear in *Kalman Filtering and Neural Networks*, chapter 7. Edited by Simon Haykin. Wiley Publishing.

[Westhoff et. al. 2004a]
D. Westhoff, T. Scherer, H. Stanek, J. Zhang and A. Knoll (2004)
"A flexible framework for task-oriented programming of service robots"
Robotik 2004 Konferenz , München. In: VDI-Bericht Nr. 1841, VDI-Verlag, Postfach 101054, 40001 Düsseldorf, Deutschland.

[Westhoff et. al. 2004b]
D. Westhoff, J. Zhang, H. Stanek, T. Scherer and A. Knoll (2004)
"Mobile Manipulatoren und ihre aufgabenorientierte Programmierung"
atp – Automatisierungstechnische Praxis 10/2004, Oldenbourg Industrieverlag GmbH, München, Germany. ISSN 0178-2320
`http://www.oldenbourg.de/verlag/at-technik/rot-atp1.htm`

[Whitney 1969]
D. E. Whitney (1969)
"Resolved motion rate control of manipulators and human prostheses,
IEEE Transactions on Man-Machine Systems, Vol. 10, p. 47-53, 1969.

[Whithey 1972]
D. E. Whitney (1972)
"The mathematics of coordinated control of prosthetic arms and manipulators
Journal of Dynamic Systems, Measures and Control, 94G(4), p. 303-309, Dec., 1972.

[Wright, Maskell and Briers 2003]
R. Wright, S. R. Maskell and M. Briers (2003)
"Comparison Of Kalman-based Methods With Particle Filters for Raid Tracking"
QinetiQ Ltd., Malvern, UK
`http://mcs.open.ac.uk/Statistics/PBS5/papers.html`

[Zhang, Hübner and Knoll 2001]
J. Zhang, K. Hübner and A. Knoll (2001)
"Learning based situation recognition by sectoring omnidirectional images for robot localisation"
Proceedings of the IEEE Workshop on Omnidirectional Vision, Budapest.

[Zhang and Knoll 1996]
J. Zhang and A. Knoll (1996)
"Constructing fuzzy controllers with B-spline models"
IEEE International Conference on Fuzzy Systems, 1996.

[Zhang and Knoll 1998]
J. Zhang and A. Knoll (1998)
"Constructing fuzzy controllers with B-spline models – principles and applications"
International Journal of Intelligent Systems, 13(2/3):257–286, February/March 1998.

Index

access point (WLAN), 227
ARCNET, 134, 142
automaton, 218
    deterministic finite automaton (DFA), 223, 236
autonomous guided vehicles (AGV), 33

behaviour based control, 110
biotechnology
    mammalian cell cultivation, 16, 19
    pilot scale cultivation, 16, 19
    sample management, 17
    short introduction, 16
borenstein tests, 50
broadcast, 237

camera, 160, 162
CCHUNK, 228, 361
cell counter, 25
centrifuge, 26
CLABDEV, 228, 229, 368
    CLABDEV, 229, 368
    CLABDEVD, 229, 368
    CLABMSG, 229, 367
CLABDEVD, 229, 368
CLABMSG, 229, 367
CNETOBJ, 228, 361, 365
Collani, Yorck von, 214
colour
    robustness, 160
    similarity, 167
    YUV colour space, 165
colour segmentation, 165
colour vision, 160
CROBOT, 231, 239, 369
CROBOTD, 232, 373

database, 224
dead reckoning (DR), 46
deterministic finite automaton (DFA), 223, 236
devices
    cell counter, 25
    centrifuge, 26
    fridge, 27
    pipetting device, 24
    sampling device, 23
DFA, 223, 236

Ferch, Markus, 214, 227
flood fill, 176
force sensor, 134
    force control, 152
fridge, 27

GenControl, 37, 90
genrob, 264
Gesellschaft für Produktionssysteme (GPS), 36
Graf, Thorsten, 227
gripper, 134
    fingers, 135

infobase, 225, 236
    infobased, 226
initial guess, 52, 188
IPA, Fraunhofer Institute, 34

kalman filter (KF), 55
    caveats, 66, 68
    extended KF (EKF), 60
    measurement model, 57
    predict and update steps, 58
    system model, 56
Kalman, Rudolph E., 56
kinematic
    mobile platform, 45
    robot arm, 133

laboratory control program (LCP), 239
LCP, 239
least squares fit (LSF), 52, 185
    initial guess, 52, 188
Lloyd, John, 137
localisation, 35, 38

alternatives, 106
borenstein tests, 50
dead reckoning (DR), 46
experiments, 111
gyro compass, 41
kalman filter (KF), 55
laser scanner, 42
odometry, 40
sensors, 39

mammalian cell cultivation, 16, 19, 239
experiments, 239
MCB, 134, 306
MHI, 133, 303
Mitsubishi Heavy Industries (MHI), 133, 303
mobile platform, 33, 36, 260
behaviour based control, 110
experiments, 111
kinematic, 45
localisation, 35, 38
motion execution, 36, 89
MP-L655, 36
path-planning, 35, 74
mobile robot system, 30
task definition, 28
motion control board (MCB), 134, 306
motion execution, 36, 89
absolute rotations, 98
absolute translations, 98
collision avoidance, 101
experiments, 111
motion types, 89
relative translations, 100
tracking, 90
velocity profiles, 92
MP-L655, 36

Neobotix, 36
GenControl, 37, 90
MP-L655, 36
PlatformControl, 91
network, 226
virtual private network (VPN), 228
wireless local area network (WLAN), 227

object displacement model, 184
object models, 181
OPERA, 214

PA-10
experiments, 147
hardware, 133, 303
joint controllers, 145
PA-Library (PA-Lib), 134, 136, 306
RCCL, 139
PA-Library (PA-Lib), 134, 136, 306
path-planning, 35
alternatives, 109
experiments, 111
map expansion, 76
maps, 74
path search, 85
tangent graph, 84
visibility graph, 82
pilot scale cultivation, 16, 19
pipetting device, 24
Poggendorf, Iris, 21, 212, 239, 255
pseudo inverse, 54

RCCL, 137, 212
RCCL++, 214
roblets, 264
robot arm, 131, 260
kinematic, 133
tool, 134
robot arm, PA-10, 133
robot tool, 309
run time type information (RTTI), 365

sample management, 17
sampling device, 23
Scheering, Christian, 226
scripts, 212, 235, 264
script language(s), 212
sensors, 39
alternatives, 104
gyro compass, 41
laser scanner, 42
odometry, 40
SICK, 42
SICK, 42
SSL, 230

tangent graph, 84
task definition, 28
TCP/IP, 226
tool, 134, 309
camera, 135, 160
force sensor, 134

gripper, 134

velocity profiles, 92
position setpoints, 95
velocity setpoints, 96
visibility graph, 82
vision, 159, 261
camera, 160
colour segmentation, 165
experiments, 190
object recognition, 177
VPN, 228, 230

WEP, 227
WLAN, 227
access point, 227
wi-fi protected access (WPA), 230
wired equivalent privacy (WEP), 227
WPA, 230

Part II

Appendices

Appendix A

Mobile Platform Documentation

This chapter gives an optional, but more verbose documentation of the mobile platform used in this work. This documentation consists of a detailed hard- and software description of the MP-L655 mobile platform as delivered by the vendor in section A.1, details about the CAN interface used to access the motors and the gyro compass in section A.2, details about the RS422 serial communication used to access the LMS-200 laser scanners in section A.3 and the application `mobiled` that implements the entire mobile platform control in section A.4.

A.1 The "Neobotix MP-L655" Mobile Platform

Although thought to be a booming market there are only very few manufacturers of mobile robots. Real World Interface (RWI), lately a subdivision of iRobot, has actually ceased the production of its well known B14 and B21 mobile robots in favour of autonomous vacuum cleaners[1]. iRobot itself, co-founded by MIT fellow Rodney Brooks[2] nowadays focuses on military applications. ActivMedia Robotics has picked up parts of the former RWI product line, selling them under new names[3]. French ROBOSOFT is the european distributor for these robots as well as some others[4].

All the products of all the above companies are mere "mobile platforms" in sometimes a very relaxed sense, and most of them are not suitable for this work simply because of their size. The MP-L655, compared to these products, has one big advantage: Neobotix offers a complete electrical and mechanical integration of the Mitsubishi PA10 robot arm and its controller, leaving only the question of the arm control software. None of the other companies offers such an integration support.

A.1.1 Hardware

The MP-L655 – see again figure 3.4 on page 37 – is built as an aluminium frame with several storeys. In the "basement" the motors and castor wheels are mounted, the first floor holds the batteries and some power electrics, the second the controlling PC and the PA10 controller and

[1] `http://www.irobot.com/rwi`, `http://www.rwii.com`.

[2] `http://www.irobot.com/corp/default.asp` `http://www.ai.mit.edu/projects/humanoid-robotics-group`.

[3] ActivMedia Robotics, 19 Columbia Drive, Amherst, NH 03031, USA. `www.activmedia.com/robots`, `http://www.activrobots.com`

[4] ROBOSOFT, Technopole d'Izarbel, F-64210 Bidart, France. `http://www.robosoft.fr`.

on the “roof” a LCD display and the PA-10 itself are mounted. The basic physical specifications are given in table A.1. Note that these dimensions do not include the mounted PA-10 robot arm – dimensions of the arm are given in chapter 4.

size:	655 × 655 mm
diameter:	844 mm
height:	560 mm
weight:	150 kg weight
payload:	app. 100 kg

Table A.1: Physical specifications of the MP-L655 (without robot arm).

The MP-L655 can be ordered with two kinematics, of which the first has been chosen for this work because a biotechnological laboratory is more likely to require complex manoeuvring than to have a very bumpy floor. The kinematics are:

1. Two spring-suspended differential drives in the rotary centre of the platform, two castor wheels in front and one castor wheel in back. This kinematic has the advantage that the platform may rotate on the spot. It has the disadvantage that a five-point-contact is not guaranteed to be stable on bumpy terrain. Also, the high number of castor wheels – if not properly oriented for the driving direction – introduce a lot of resistance that has to be overcome, which means that on a suboptimal floor the drive wheels are more likely to slip.

2. Two differential drives at the front of the platform and one castor wheel in back. This kinematic has the advantage that a three-point-contact is always stable and that no spring suspension is needed. It has the disadvantage that the platform cannot rotate on the spot and therefore needs more free space to perform certain manoeuvres and therefore a more complex motion planning.

The electrical parameters of the MP-L655 are given in table A.2. The main voltage of 48 V is used to drive the motors, the on-board PC and the robot arm. The total capacity of 3840 Wh allows up to 12 h of operation with moderate use of the arm and/or the platform. Auxiliary voltages are available for additional user equipment.

The platform does not come with an automatic charging station, therefore a custom solution as in figure A.1 has been built. This charging station allows the robot to safely charge the batteries during the idle time. Its copper contacts are isolated from the operating voltage unless pressed into the case against a spring, so that accidental and/or unnoticed bumping into a person or a person unintentionally playing with the contacts can cause no harm.

The drive motors are a highly integrated proprietary Neobotix’ development. They combine a sine wave commutated brushless servo drive, a backlash-free gear box, high resolution wheel encoders (4096 increments / motor revolution), an electromagnetic brake and a Siemens C164 microcontroller and the necessary power electronics in a single case as in figure A.2. The C164 basically runs a speed control loop for the motor, interfaces to the brake and implements a few safety constraints like checking for the motor temperature. With these motors the platform can reach a maximum rotational speed of 90 °/s and a maximum translational speed of 1 m/s.

batteries:	8 × 12 V, 40 Ah lead-acid
main voltage:	48 V (4 × 12 V in a row)
total capacity:	80 Ah (2 × 40 Ah parallel)
total power:	3.84 kWh
auxiliary voltages:	+24, +12 and +5 V

Table A.2: Electrical specification of the MP-L655.

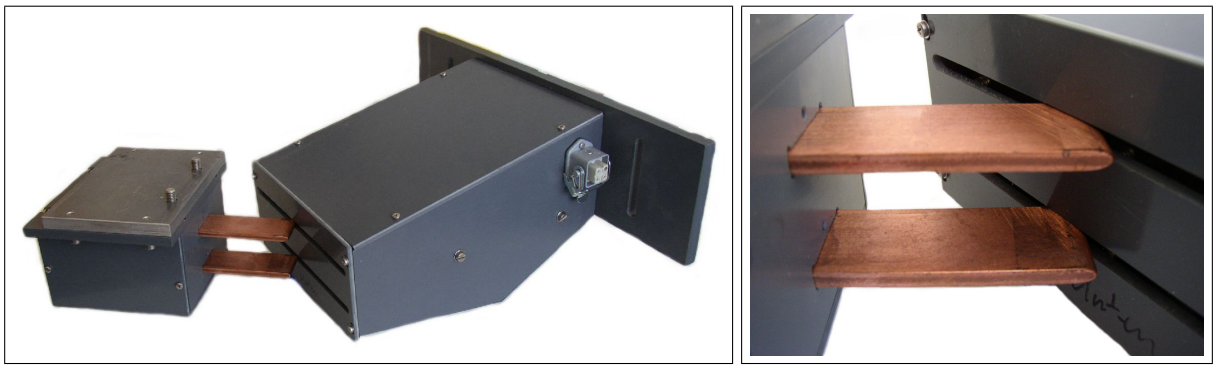


Figure A.1: The contact mechanism used for the automatic charging station.

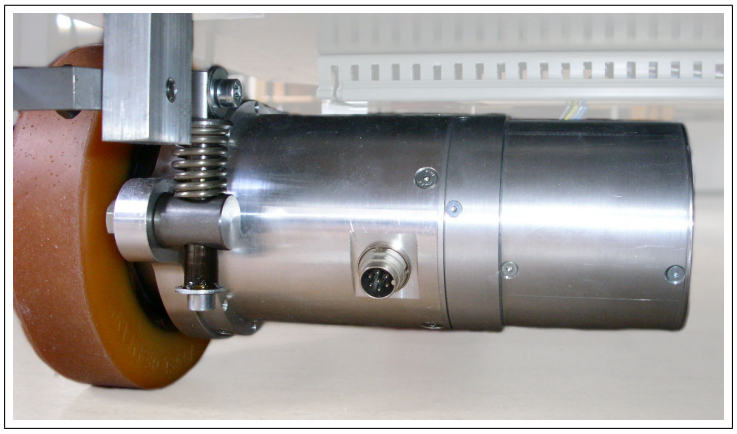


Figure A.2: A drive motor, integrating the actual motor, wheel encoders, a gear box, power electronics and the controller.

In addition to the odometry sensors provided by the motor’s C164 microcontroller the MP-L655 used in this work has an electrical gyro compass and two SICK laser scanners, making for a total sensor set as in table A.3. The gyro compass is connected to a third microcontroller, a Siemens C167, on an I/O board that is also used to read out a joystick interface and the main battery voltage. Other versions of the MP-L655 add an ultrasonic sensor board with yet another C167 microcontroller, and low-level bumper bars. These additional sensors are required to gain approval of official safety regulations[5].

[5]In Germany there is not yet an explicit class of safety regulations for mobile robots because they are still not very common, so standard safety regulations from other machinery are applied instead. These regulations for example state that a system must have a low-level emergency stop mode that does not rely on high-level software (that may have crashed). A plain SICK laser scanner as used on the MP-L655 is therefore not enough because it can be operated in either a mode for measuring distances and reflectance or a mode for monitoring safety areas and actively signalling violations on an I/O bit, but not in both simultaneously. See also section 3.2.1.3.

2 × odometry sensors (1 per motor)
1 × electrical gyro compass
2 × laser scanners (1 in front, 1 in rear)

Table A.3: MP-L655 sensors.

An on-board PC in a small industrial case is used to implement the control program by accessing the other components. The PC is connected to the C164 and C167 controllers via a high speed CAN bus – whose protocol is documented in the appendices in section A.2 – and to the laser scanners via RS422 serial lines (see again figure 3.5. on page 38). Due to the large battery capacity the PC is not necessarily required to use power-saving equipment, therefore it can be built of virtually any standard hardware. Since in this work the PC must also control a robot arm with its sensors it consists of the following hardware:

- An *ICP Industrial Computer Products*[6]. “ROCKY-4784EV” CPU backplane with a 2 GHz Intel Pentium 4 processor, 256 Mb RAM, a 2 1/2 inch notebook harddisc[7] with 30 Gb, built-in graphics and (wired) ethernet network,
- an *ATI Industrial Automation*[8] (ATI-IA) ISA board to connect to the force/torque sensor on the robot arm,
- a *Moxa Technologies*[9] CP-132 PCI board with two RS422 serial ports to connect to the SICK laser scanners (see subsection 3.2.1.3),
- a *Matrox*[10] “Meteor” PCI framegrabber board to access the camera on the robot arm (see chapter 5 for details),
- a (modified) *SOHARD*[11] “SH ARC-PCI” PCI arcnet board to access the robot arm itself (see section 4.3.2 for details) and
- an *ELSA / Lancom*[12] “Airlancer PCI-11” PCI board to interface to an “Airlancer MC-11” PCMCIA IEEE 802.11b wireless LAN card to connect the platform with the main network.

A.1.2 Software

The platform comes with a proprietary control software that is supposed to handle all the aspects of localisation, path-planning and motion execution that are needed for controlling a mobile platform. This software still has to be considered “under development” because since the time

[6] ICP Deutschland GmbH, Humboldtstr. 36, D-70771 Leinfelden-Echterdingen, Germany, `http://www.icp-deutschland.de`

[7] Notebook harddiscs are more rugged than ordinary desktop harddiscs in terms of shock-resistance, which is necessary because of the platform driving over potentially bumpy floors.

[8] c/o SCHUNK GmbH & Co. KG, Bahnhofstraße 106-134, D-74348 Lauffen am Neckar, Germany, `http://www.schunk.de`, `http://www.ati-ia.com`

[9] Moxa Germany, Heinrich-Lanz-Straße 4, D-69514 Laudenbach, Germany, `http://www.moxatech.de`

[10] Matrox Electronic Systems GmbH, Unterhaching, Germany, `http://www.matrox.com`

[11] SOHARD AG, Würzburger Straße 197, D-90766 Fürth, Germany, `http://www.sohard.de`

[12] LANCOM Systems GmbH, Adenauerstraße 20 / B2, D-52146 Würselen, Germany, `http://www.lancom-systems.de`

from the purchase of the platform it has undergone major changes. The history of these versions is important because it explains why an in-house replacement had to be developed.

The original software called GENCONTROL by Neobotix/GPS is a large, monolithic, self-contained Windows C++ application, a screenshot of which can be seen in figure A.3. It uses a map of known laser reflector markers, the gyro compass and the odometry sensors for localisation. It also has a map of polyline obstacles, but does not use it for path-planning. Instead it uses an obstacle avoidance mode to drive around corners to reach a target without bumping into walls (or other obstacles). This of course at most only works with targets that are quite trivially reachable around just one corner, yet it has been observed to fail more than once. For GENCONTROL to be used in a maze-like environment an external path planner therefore has to be implemented.

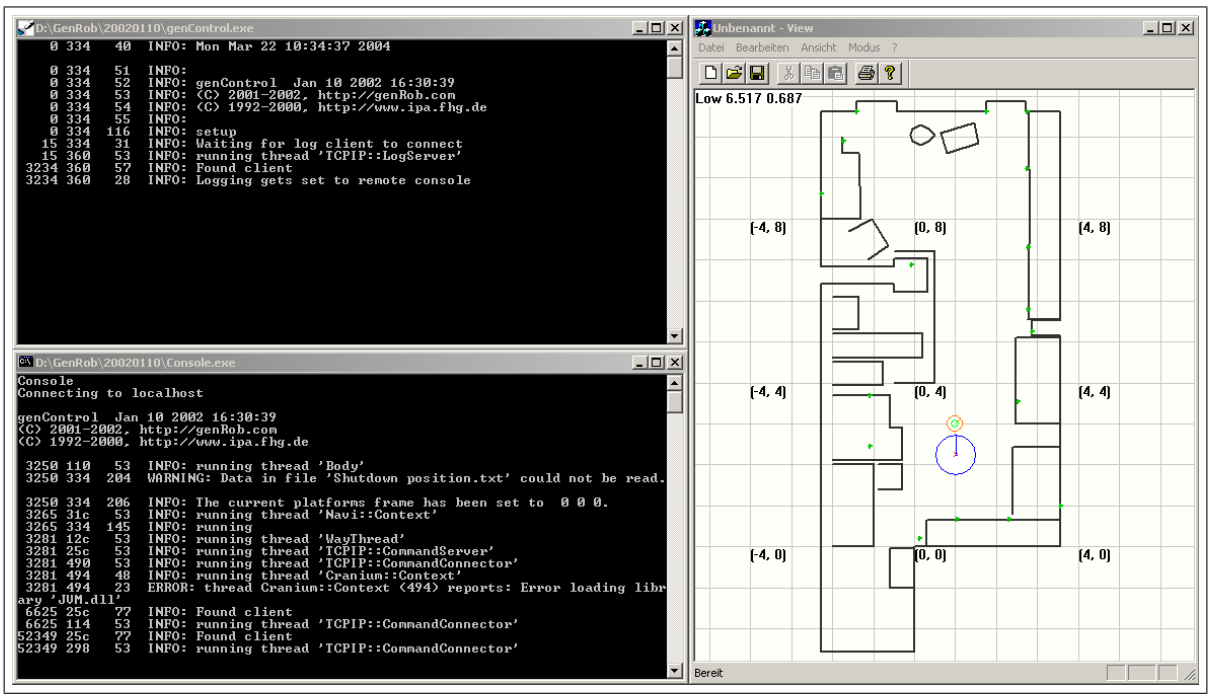


Figure A.3: Neobotix’ GENCONTROL software.

The interface to GENCONTROL’s functionality is a TCP/IP socket running a simple ASCII protocol giving access to the sensors and motion primitives. Using this interface the platform can be initialised, different operating modes (standby / target / joystick) can be selected, motions can be issued to targets given as (x, y, α)-triplets and the laser scanners can be read out to visualise the environment and verify the reported platform position.

One major problem of GENCONTROL is that it has no concept of a “motion” as an entity that starts, runs and is definitely finished at some time. The way GENCONTROL manoeuvres is that it constantly and indefinitely *tracks* a target (that can be changed at any time). This tracking is divided into three phases, in which the platform rotates towards the target, drives to the target position and then rotates to the desired target orientation.

However, on the one hand side it is not possible to read out which phase GENCONTROL is in and on the other hand side it can jump back from phase 3 to phase 1 if it decides that rotating towards the target orientation has brought it too far away from the target position (for example

because of a slippery ground). Therefore it is highly non-trivial to check if the platform has arrived at the target or not.

The only way this simple question can be answered is by checking if it has come within a certain radius of the target and has no longer been moving the wheels for a certain time, and even this can fail in case of an emergency stop because of an obstacle close to the target. Besides that, determining the catch radius is non-trivial because the localisation is very noisy.

Measurements done by A. Schneider and D. Westhoff in [Schneider and Westhoff 2002] show that GENCONTROL’s localisation and motion control are not accurate and stable enough to meet the requirements of this work.

On the one hand side the platform can get trapped in endless corrective motions when trying to reach a target with a desired precision that is lower than the localisation noise. On the other hand side the worst deviations observed are that the platform deviates from the target by some 5 cm in one direction, but reports to have deviated 5 cm in the exact opposite direction.

Overall, these effects are not tolerable. Since no source code for GENCONTROL was available to fix these problems and since additional external software (a path planner) was needed anyway, an in-house replacement had to be built.

Since then Neobotix has released a new software called PLATFORMCONTROL , a screenshot of which can be seen in figure A.4. This software has been improved in several aspects. It is a complete rewrite incorporating latest Fraunhofer IPA research results (including wall recognition) and is separated into a Windows C++ controller application (PltfCtrl) and a JAVA *graphical user interface* (PltfCtrlGui). Neobotix claims that the localisation and positioning are much better than with the old software, but since the new one was only released after the solution presented in this work was completed it has not been further investigated.

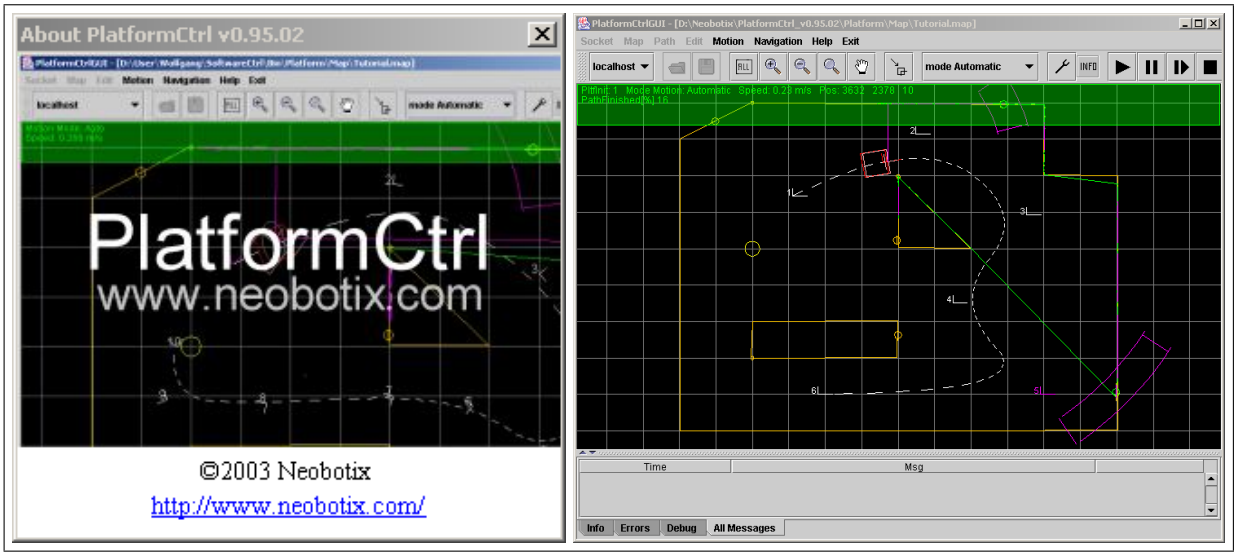


Figure A.4: Neobotix’ PLATFORMCONTROL Software.

One final argument against using any of these commercial solutions has to be emphasised: Both GENCONTROL and PlatformControl are – though having a platform independent Java user interface – at their core Windows applications and need a Windows operating system. Using any of these programs means to have to install a second PC on the platform because the arm control

software runs under the Linux operating system. Such a second PC would be very troublesome because of the limited space available on the mobile platform.

A.2 The CAN Interface

The platform's low-level interface is a high-speed *controller area network* (CAN) that is attached to the on-board PC (see again figure 3.5 on page 38). This CAN offers access to the C164 motor controllers with their wheel encoders and the C167 I/O-controller with the gyro compass. The CAN interface is proprietary by Neobotix and only documented by means of header files. Therefore, this section separately documents the CAN hardware, packet format and commands used to access the individual controllers.

A.2.1 CAN Hardware

The CAN is a serial fieldbus developed by the *Robert Bosch GmbH* in the 1980s as a communication system to replace explicit cabling for the increasing number of actors, sensors and controllers in the car industry. It supports data rates up to 1 MBps and is very robust against electrical and/or magnetic disturbances. Many companies offer a wide range of CAN hardware, taking advantage from the fact that CAN is an open standard (ISO 11898 for data rates up to 125 kbps and ISO 11519 for data rates above 125 kbps) not requiring any licensing fees.

On the MP-L655 the CAN is connected to the PC by a *PEAK System Technik GmbH*[13] "PCAN-Dongle". This dongle is attached to the parallel port and therefore does not require a free slot on the PC – a fact that is quite important in a setup of miniaturised hardware. It is built around a PHILIPS CAN-Controller SJA1000 with 16 MHz clock frequency and compliant to CAN specifications 2.0A (11-Bit ID) and 2.0B (29-Bit ID). What makes the dongle interesting is that it comes with a well-documented Linux driver providing real `/dev/pcan*` devices.

A.2.2 CAN Frames

The CAN is not just an electrical specification for transmitting raw bits like a RS232 serial line, but also a protocol specification defining several frame types. The most commonly used frame type is the normal data frame, consisting of some flags, a device address (ID), a payload size field and the payload itself. The PCAN-Dongle driver abstracts from the protocol details so that the user only has to supply a target ID and the payload data.

Technically, the CAN is a *broadcasting* medium, that means physically each node sees all messages. It is in the responsibility of the designer of the bus to manually assign each node a unique ID and to manually filter out packets not meant for this node. The IDs can be either 11 bit (CAN 2.0A) or 29 bit (CAN 2.0B). On the MP-L655 the device addresses are as in table A.4.

Due to the intended real-time capabilities in combination with the (compared to typical ethernet) low data rate the payload is limited to up to 8 bytes of user data per frame. This limited payload is not really a restriction since the CAN usually controls low-level devices which only

[13] PEAK System Technik GmbH, Otto-Röhm-Straße 69, D-64293 Darmstadt, Germany, `http://www.peak-system.com`

address	controller	direction		
0x101	C167 i/o	PC	→	CTRL
0x100	C167 i/o	CTRL	→	PC
0x201	C164 right motor	PC	→	CTRL
0x200	C164 right motor	CTRL	→	PC
0x301	C164 left motor	PC	→	CTRL
0x300	C164 left motor	CTRL	→	PC
0x401	C167 ultrasonic	PC	→	CTRL
0x400	C167 ultrasonic	CTRL	→	PC

Table A.4: The CAN device IDs used on the MP-L655. The IDs concerning the ultrasonic board are only mentioned for completeness – they are not used on the platform in this work because it does not have an ultrasonic board.

need to communicate very few bytes. On the MP-L655 all data frames have 8 bytes, padding unused bytes with zeros.

The SJA1000 used in the PCAN-Dongle is capable of buffering up to 8 CAN frames, making it quite improbable that a message should get lost. In case it does, the protocol carried within the payload data has to ensure that no harm is done. On the MP-L655 the commands have been designed to minimise traffic and so that it does not hurt if a reply should get lost.

In the following the general layout of the CAN frames on the MP-L655 and some selected commands needed to access the C164 motor controllers and the C167 I/O controller are given. Since these do have changed in the past this description has to be taken as strictly inofficial. For any official documentation on these, unlisted or new commands the Neobotix documentation has to be used, available at the Neobotix website[14].

The general frame format for *commands* on the MP-L655 is given in table A.5, where CMD is the command value.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
CMD	...	...	...	...	...	...	...

Table A.5: General CAN user data layout for command packets on the MP-L655.

The general frame format for *replies* to a command on the MP-L655 is given in table A.6, where CMD is the command value as sent in the command and STATUS is a 2-bit code defined as the enumeration:

```
enum Ser_Can_Msg
{
  Msg_OK,
  Msg_Error,
  Msg_NotAccept,
  Msg_NoAction
};
```

[14] `http://www.neobotix.de/downloads`

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
...	...	...	...	...	...	ID	(CMD << 2) | STATUS

Table A.6: General CAN user data layout for reply packets on the MP-L655.

The variables denotes in this table are to be understood as follows: "FOO" stands for an up to 8-bit value that fits into one byte and "FOO_0^7" stands for (in this case) only the bits 0 to 7 of a larger value. Variables larger than 8 bit are encoded in a *big-endian* style.

A.2.3 The C164 Motor Controller(s)

Each motor has its own C164 controller integrated and can be accessed separately with a set of commands, which in theory makes it possible to have more than two motors on the platform. The commands are defined in the Neobotix header file `Cmd_MotCtrl.h` as an enumeration `Cmd_MotCtrl`, an excerpt of which will be explained here.

```
enum Cmd_MotCtrl
{
  CMD_MOTCTRL_CONNECT,
  CMD_MOTCTRL_DISCONNECT,
  CMD_MOTCTRL_SETDIGOUT,
  CMD_MOTCTRL_GETDIGIN,
  CMD_MOTCTRL_GETANALOGIN,
  CMD_MOTCTRL_SETMOTIONTYPE,
  CMD_MOTCTRL_GETPOSVEL,
  CMD_MOTCTRL_GETSTATUS,
  CMD_MOTCTRL_DISABLEBRAKE,
  CMD_MOTCTRL_ENABLEMOTOR,
  CMD_MOTCTRL_SYNCHMOTOR,
  CMD_MOTCTRL_ENABLECOMM,
  CMD_MOTCTRL_SETCMDVAL,
  CMD_MOTCTRL_SETCTRLPARA,
  CMD_MOTCTRL_GETCTRLPARA,
  CMD_MOTCTRL_SETPOSCTRL,
  CMD_MOTCTRL_GETPOSCTRL,
  CMD_MOTCTRL_SETEMSTOP,
  CMD_MOTCTRL_RESETEMSTOP,
  CMD_MOTCTRL_ERROR_STOPMOTION,
  CMD_MOTCTRL_UNKNOWN
};
```

`CMD_MOTCTRL_CONNECT` is used to initially connect to the controller. This is more a logical initialisation than a physical one because technically the communication partners are already connected. It basically means that the controller enters a state where it accepts other commands. `CMD_MOTCTRL_CONNECT` does not carry any parameters and triggers a reply with no parameters.

Byte 1
CMD_MOTCTRL_CONNECT

CMD_MOTCTRL_DISCONNECT is used to terminate a "connection". After this commands the controller will no longer accept any other command except a new CMD_MOTCTRL_CONNECT. CMD_MOTCTRL_DISCONNECT does not carry any parameters and does not trigger a reply.

Byte 1
CMD_MOTCTRL_DISCONNECT

CMD_MOTCTRL_DISABLEBRAKE is used to control the mechanical brake in the motors. This brake is enabled by default (fail-safe) and has to be actively released by an electromagnet. This magnet consumes about 1 A currency at 12 V, so it is a sound idea to re-enable brakes whenever the platform is not used. CMD_MOTCTRL_DISABLEBRAKE carries a 1-bit true/false FLAG as parameter and triggers a reply with no parameters.

Byte 1	Byte 2
CMD_MOTCTRL_DISABLE_BRAKE	FLAG

CMD_MOTCTRL_ENABLEMOTOR is used during initialisation to enable (disable) the power electronics of the motor. It does not yet mean that the motor can be used afterwards – that requires syncing the motor. CMD_MOTCTRL_ENABLEMOTOR carries a 1-bit true/false FLAG and an 8-bit TORQUE value as parameters and triggers a reply with no parameters.

Byte 1	Byte 2	Byte 3
CMD_MOTCTRL_ENABLEMOTOR	FLAG	TORQUE

CMD_MOTCTRL_SYNCHMOTOR is used during initialisation to synchronise the motor, i.e. to find the zero index of the wheel encoder. The motor is still not usable after syncing – the commutation still needs to be enabled. CMD_MOTCTRL_SYNCHMOTOR carries an 8-bit TORQUE value as parameter and triggers a reply with the expected and measured 16-bit positions of the encoder zero index (only needed for hardware debugging).

Byte 1	Byte 2
CMD_MOTCTRL_SYNCHMOTOR	TORQUE

Byte 1	Byte 2	Byte 3	Byte 4
POSMEAS_8^{15}	POSMEAS_0^7	POSEXP_8^{15}	POSEXP_0^7

CMD_MOTCTRL_ENABLECOMM is used during initialisation to enable the commutation of the motor, which is the final step to start it. CMD_MOTCTRL_ENABLECOMM carries no parameters and triggers a reply with no parameters.

Byte 1
CMD_MOTCTRL_CONNECT

CMD_MOTCTRL_GETSTATUS is used to request various state information about the controller. It can be called at any time after connecting to a controller, even when the motor is not yet initialised. `CMD_MOTCTRL_GETSTATUS` carries no parameters and triggers a reply with a 16-bit STATE information (consisting of bits as defined below in `enum STATE_Ctrl`) and the measured 16-bit TEMP(erature) (useful for detecting thermic overloads, but needs conversion).

Byte 1
CMD_MOTCTRL_GETSTATUS

Byte 1	Byte 2	Byte 3	Byte 4
$STATE_8^{15}$	$STATE_0^7$	$TEMP_8^{15}$	$TEMP_0^7$

```
enum STATE_Ctrl
{
  ST_Ctrl_LimSwHardPosMax = 1,
  ST_Ctrl_LimSwHardPosMin = 2,
  ST_Ctrl_LimSwSoftPosMax = 4,
  ST_Ctrl_LimSwSoftPosMin = 8,
  ST_Ctrl_TempMax = 16,
  ST_Ctrl_EMStop = 32,
  ST_Ctrl_Watchdog = 64
}
```

CMD_MOTCTRL_GETPOSVEL is used to request the current motor position and velocity. The position is an absolute value measured between the motor and the gear box. `CMD_MOTCTRL_GETPOSVEL` carries no parameters and triggers a reply with a 32-bit ANGLE[15] and a 16-bit VEL(ocity) value.

Byte 1
CMD_MOTCTRL_GETPOSVEL

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
$ANGLE_{24}^{31}$	$ANGLE_{16}^{23}$	$ANGLE_8^{15}$	$ANGLE_0^7$	VEL_8^{15}	VEL_0^7

CMD_MOTCTRL_SETCMDVAL is used to send new target values to the motor. `CMD_MOTCTRL_SETCMDVAL` carries 16-bit VEL(ocity) and 32-bit POS(ition) values (current implementations only use the velocity values, the position values are reserved for future use for a more precise fine-positioning of the motor) and triggers two replies, one of type `CMD_MOTCTRL_GETPOSVEL` and one of type `CMD_MOTCTRL_GETSTATUS`. This is deliberate to minimise the traffic on the bus by avoiding sending commands which can be considered unnecessary because it can be predicted that the user is going to call them.

[15] Current versions have a known bug that this value wraps near 0x08000000, see the source code for details.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CMD_MOTCTRL_SETCMDVAL	POS_{24}^{31}	POS_{16}^{23}	POS_{8}^{15}	POS_{0}^{7}	VEL_{8}^{15}	VEL_{0}^{7}

A.2.4 The C167 I/O Controller

The I/O controller is a C167 mainly interfacing to the gyro compass, the joystick and the battery voltage meter. The commands are defined in the Neobotix header file `Cmd_IOBoard.h` as an enumeration `CMD_IOBOARD`, an excerpt of which will be explained here.

```
enum CMD_IOBOARD
{
  CMD_IOBOARD_CONNECT,
  CMD_IOBOARD_DISCONNECT,
  CMD_IOBOARD_GETDIGIN,
  CMD_IOBOARD_SETDIGOUT,
  CMD_IOBOARD_GETANALOGIN,
  CMD_IOBOARD_GETGYROVAL,
  CMD_IOBOARD_ZEROGYRO,
  CMD_IOBOARD_GETJOYVAL,
  CMD_IOBOARD_GETVBATT,
  CMD_IOBOARD_GETSTATUS,
  CMD_IOBOARD_SETCTRLPARA,
  CMD_IOBOARD_GETCTRLPARA,
  CMD_IOBOARD_UNKNOWN
};
```

`CMD_IOBOARD_CONNECT` is used to "connect" to the controller. The same comments as for the motor controllers apply. `CMD_IOBOARD_CONNECT` carries no parameter and triggers a reply with no parameters.

Byte 1
CMD_IOBOARD_CONNECT

`CMD_IOBOARD_DISCONNECT` is used to "disconnect" from the controller. The same comments as for the motor controllers apply. `CMD_IOBOARD_DISCONNECT` carries no parameter and does not triggers a reply.

Byte 1
CMD_IOBOARD_DISCONNECT

`CMD_IOBOARD_GETSTATUS` is used to read the combined data of the sensors attached to the I/O controller, thus saving bandwidth because the commands do not have to be sent separately. `CMD_IOBOARD_GETSTATUS` carries no parameters and triggers two replies, one of type `CMD_IOBOARD_GETSTATUS` and one of type `CMD_IOBOARD_GETDIGIN` (not described here). The `CMD_IOBOARD_GETSTATUS` reply holds two 8-bit values for the joystick readings, a 16-bit ANGLE value for the gyro (see `CMD_IOBOARD_GETGYROVAL`) and a 16-bit value for the battery voltage (see `CMD_IOBOARD_GETVBATT`).

Byte 1
CMD_IOBOARD_GETSTATUS

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CMD_IOBOARD_GETSTATUS	JOYX	JOYY	$ANGLE_8^{15}$	$ANGLE_0^7$	$VOLTAGE_8^{15}$	$VOLTAGE_0^7$

CMD_IOBOARD_ZEROGYRO is used to initialise the gyro compass (set it to zero). This is – strictly speaking – not necessary because only differences of gyro values are used for the localisation, but may be helpful. CMD_IOBOARD_ZEROGYRO carries no parameters and triggers a reply with no parameters.

Byte 1
CMD_IOBOARD_ZEROGYRO

CMD_IOBOARD_GETGYROVAL is used to read the gyro compass value. This value is an absolute position value integrated over the gyro signal by the controller. To obtain the velocity this value has to be differentiated again. CMD_IOBOARD_GETGYROVAL carries no parameters and triggers a reply with a 16-bit ANGLE value (needs to be converted).

Byte 1
CMD_IOBOARD_GETGYROVAL

Byte 1	Byte 2
$ANGLE_8^{15}$	$ANGLE_0^7$

CMD_IOBOARD_GETJOYVAL is used to read out the joystick readings. CMD_IOBOARD_GETJOYVAL carries no parameters and triggers a reply with two 8-bit readings, the x- and y-axis of the joystick.

Byte 1
CMD_IOBOARD_GETJOYVAL

Byte 1	Byte 2	Byte 3
CMD_IOBOARD_GETJOYVAL	JOYX	JOYY

CMD_IOBOARD_GETVBATT is used to read the platform battery voltage. CMD_IOBOARD_GETVBATT carries no parameters and triggers a reply[16] with a 16-bit VOLTAGE value (needs to be converted).

Byte 1
CMD_IOBOARD_GETVBATT

Byte 1	Byte 2
$VOLTAGE_8^{15}$	$VOLTAGE_0^7$

[16] Older versions tagged this reply as CMD_IOBOARD_GETJOYVAL.

A.3 The LMS-200 Serial Communication

Communication with the SICK LMS-200 laser scanners is done via RS422 serial lines according to a protocol defined in the *telegram definition manual* (TDM)[17]. Both the hard- and software aspects of this communication do have their originalities which must be observed in order to avoid serious problems.

A.3.1 Serial Communication Hardware Issues

According to the TDM the scanners support data rates of 500000, 38400, 19200 and 9600 bps. In the same manual a list of values can also be found which are returned when querying the scanner for its internal status ([SICK AG 2000], `0xB1` reply). These values lead to the idea that they are used as divisors for a frequency of 1 MHz to generate the data rates as in table A.7, leading to a (neglectable) error in the lower rates and a totally uncommon highest data rate that can only be explained by SICK's desire to save costs at all price and not install a simple additional quartz oscillator.

speed	flag	divisor	real speed	error
9600	0x8067	103	9708	1.13 %
19200	0x8033	51	19607	2.12 %
38400	0x8019	26	38461	0.16 %
500000	0x8001	2	500000	0.00 %

Table A.7: Baud rates as derived from [SICK AG 2000]. The manual only lists the *flag*s and not the formula, but it can be guessed as 1000000 / (flag - 0x7fff). The resulting error in the lower rates is small enough to not irritate a receiver that is running on the correct speed because it will re-sync on the rate after each byte.

On the PC architecture the *universal asynchronous receiver / transmitter* (UART) was historically driven with a frequency of 1.8432 MHz, which the chip immediately divided by 16 to yield a *baud base* of 115200 bps. It is this baud base which is further divided to yield the serial data rate, the original 8250 UART only allowing divisors of 2 and therefore at most 57600 bps. Modern chips like the 16550 UART can be driven with a much higher frequency, most commonly 14.7456 MHz = 8 · 1.8432 MHz, leading to eight times the baud base (921600 bps = 8 · 115200 bps) and therefore eight times the data rate as for the same divisor as with the original frequency. For compatibility with the original rates these chips have an additional internal divide-by-8 logic that can be activated and deactivated according to the desired rate. If in high-speed mode, a divisor of 2 would therefore lead to a data rate of 460800 bps, which is quite common.

However, a data rate of 500000 bps can not be achieved by any integer divisor using this frequency, so a customary UART like the typical on-board chips can **not** be used to interface to the LMS-200. Instead the PC is equipped with a *Moxa*[18] "CP-132" PCI board with an 16550 UART on which the 14.7456 MHz quartz has been exchanged by a 16 MHz one. Using this quartz the

[17] See [SICK AG 2000].

[18] Moxa Germany, Heinrich-Lanz-Straße 4, D-69514 Laudenbach, Germany, `http://www.moxatech.de`

board actually matches all the LMS-200 data rates without error. The resulting integration of the scanners can be seen in figure A.5.

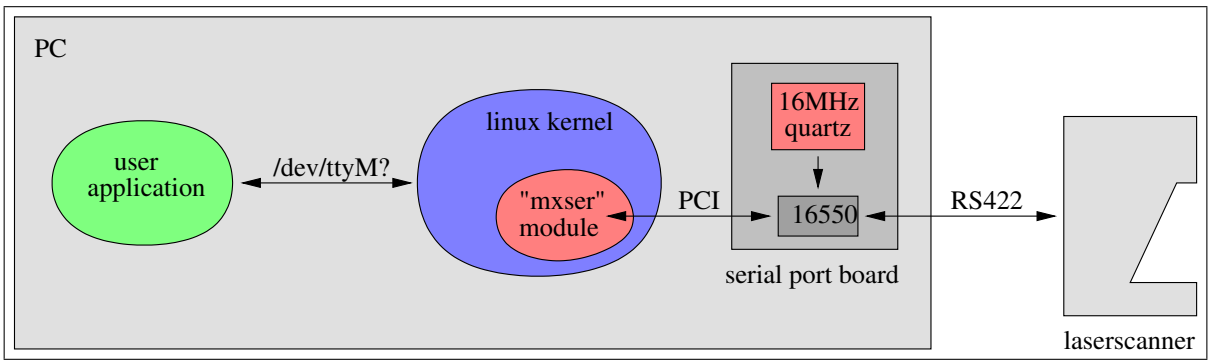


Figure A.5: The integration of the SICK laser scanner into the Linux kernel involves several questions: Due to the uncommon transfer speed the clocking quartz on the serial port has to be modified and a special kernel module has to be used that knows about this.

Adding to the above complications there exists yet another potential snag: The 16550 on the Moxa board has a 16 byte FIFO buffer to avoid serious overrun problems, but its interface to the PCI bus is rather slow. It has to be accessed using explicit I/O commands, reading each byte separately. Tests have shown that a read of a single byte off the chip's FIFO can take considerably more than 1 us. The practical upshot of this is that if one uses the FIFO buffer to avoid a high interrupt load the interrupt routine may not be fast enough to empty the buffer and overruns may in fact occur, in particular if two 16550 are receiving data from two laser scanners simultaneously. In order to avoid this the FIFO setting *must not* be set to their maximum value, but even then occasional overruns do occur. Other variations of UARTs like the yet newer 16650 and 16750 have a larger FIFO (32 and 64 bytes) but are unlikely to have a better bus interface, though due to a lack of available boards this has been left untested. What remains is that for two active scanners the CPU of the on-board PC reading the data is under a load of 20-30 % in an interrupt routine just for reading data off the UART, regardless of the speed of the CPU

A.3.2 Serial Communication Software Issues

Another thing that has next to be taken care of is the software driver. Moxa ships a Linux driver with the board that allows to use the higher data rates, but of course this driver does not know about the modified quartz. Basically, in order to use it the divisor tables have to be changed to reflect the higher baud base. The only thing that remains after these changes is that Linux does not allow to use *baud rate* values when setting the data rates, but *enumeration flags* which reflect the underlying divisors. So in order to switch to 500000 bps a constant called `B460800` has to be used, and it is up to the user to not get irritated by this.

Yet, after the hardware connection is properly established, other problems remain. The TDM specifies that

- the maximum time between two bytes of a telegram received from the LMS-200 may be up to 14 ms
- the maximum time between two bytes of a telegram send to the LMS-200 must not be longer than 6 ms, because otherwise a timeout is assumed and the telegram discarded

- the minimum time between two bytes of a telegram send to the LMS-200 must not be shorter than 55 μs.

The first condition is trivial to meet: A reading process simply has to ignore all timeout considerations and just read data as it arrives. If there should have been a gap in the data stream because a byte was dropped this can always be detected with a checksum later on.

The second condition is not that trivial, but still easy: Knowing that data is send in an interrupt routine once buffered by the system it is just almost impossible that a gap of 6 ms should occur, the more because of the UART's FIFO. Such a gap could only occur if some other interrupt routine would block the processor for more than that amount of time, and this should not be the case with Linux - though it is not proven.

The last condition, however, does present a problem because the normal spacing at 500000 bps would be 20 μs (8 data bits, 1 start and 1 stop bit). Requiring a 55 μs spacing means that the data cannot be send as a continuous stream, which at least means that it cannot be given to the Linux kernel with one large `write()` call because the timing of what happens after the `write()` is out of the hands of the application. What can be done instead is sending the data byte by byte using `write()`s of only 1 byte and manually running a delay loop in between. Such a delay loop would of course execute in the user domain and be subject to all scheduling effects that may occur. Even if it is running in a tight loop not giving up the processor it may be preempted for other tasks, leading to that larger gaps in the stream may occur. In the worst case it might become possible that the second condition gets violated this way.

Overall, it cannot be guaranteed that sending telegrams to the LMS-200 is successful on a multitasking operating system, and so a close watch on timeouts during an attempt to receive a reply is needed to detect sending problems. Fortunately the only places where telegrams have to be send is during initialisation. Once initialised the LMS-200 is put into a mode where it continuously sends scan telegrams without any further need to talk to it.

A.3.3 Telegrams and Modes

Once all low-level communication issues are resolved telegrams can be sent to the scanner for execution. Each telegram consists of a header, a command byte, additional command data and a checksum as in table A.8. 16-bit values are encoded in the *little-endian* (Intel) format, the lowest significant byte first. Upon receiving a telegram the scanner is expected to reply with either an `ACK` (`0x06`) or a `NACK` (`0x15`) within 60 ms (after a `NACK` the host should wait at least 30 ms before retrying). This only indicates whether a telegram has been received correctly, not what the status of execution of that command is. Depending on the command it may still take up to some seconds until the scanner sends a reply telegram.

name	bytes	explanation
STX	1	0x02 start byte
ADDRESS	1	0x00..0x7f host → LMS, +0x80 LMS → host
LENGTH	2	DATA length (not including checksum)
CMD	1	0x00..0x7f command byte, +0x80 reply
DATA	LENGTH-1	arbitrary command data
CRC	2	CRC checksum from STX to DATA

Table A.8: General telegram format on the LMS-200

Using these telegrams the scanner may be sent commands to execute, for example the RESET command (`0x10` telegram) to be initialised to a known state[19]. This rises the problem that in order to successfully send a RESET command one already has to know the transmission speed the scanner is running on, which is not necessarily the case. The initialisation routine therefore first tries to abort whatever command the scanner is working at and set its speed to 9600 bps by sending it an appropriate SETSPEED command (`0x20` telegram with `0x42` mode). Since the actual transmission speed is yet unknown no return errors can be evaluated and the procedure has to be repeated at all possible transmission speeds of 9600, 19200, 38400 and 500000 bps, one of which the scanner will be running at. After this the RESET command may be send, this time evaluating a return error. Once the state of the scanner is established it may be set to 500000 bps again (`0x20` telegram with `0x48` mode) and finally used.

The scanner has several modes in which it uses different resolutions and does or does not report reflection values as real reflection values or just a "blending" flag. Since the data field in the reply telegrams has limited bits the distance resolution is linked to the maximum distance value. They can the changed from a range of 8, 16 and 32 m with millimetre resolution up to 80 m with centimetre resolution. A decimetre resolution is also available, but it does not enlarge the operating range any further. Also, the angular resolution can be changed to either 0.25, 0.5 and 1 °. In 1 and 0.5 ° mode the scanner can report all 181 or 361 scan values in one telegram, but in 0.25 ° mode it can not. Instead it replies with several telegrams in which the data is delivered in what SICK calls an "interlaced mode", because it has to acquire them using several rotations of the mirror. The details of all possible modes and their data format can be found in the TDM.

On the MP-L655 the scanners are operated in a *continuous mode* (`0x20` telegram with `0x24` mode). In this mode it continuously delivers 361 distances of up to about 8 m with millimetre resolution and an angular resolution of 0.5 °. The reply telegram format for this continuous mode is given in table A.9. It is because of this mode that overruns in the serial communication are not really fatal: If an overrun occurs the telegram is discarded due to its broken checksum, but another telegram will soon follow anyway. It just means that the localisation will have to skip one cycle, which is not a problem unless it happens too often and/or in succession, which experience has shown is not the case.

Since the mirror takes 26 ms for one scan the scan frequency can be as high as 38 Hz. Given that each scan telegram consists of 732 bytes this results in an effective data flow of only up to 27 kBytes/s, or almost 280 kBit/s. The physical transmission rate of 500000 kbps is not even barely touched, so all the hardware efforts to get a communication going at this troublesome rate is actually wasted.

A.3.4 Summary

Summing up the difficulties of establishing a reliable serial communication in the first place and considering the amount of processor load this communication introduces, using a RS422 serial line has to be considered a bad idea. Seen from an efficiency point of view a more modern transmission media like USB with a packet oriented approach would be highly desirable, but since the LMS-200 is often used in rough industry environments the robustness advantages of RS422 may have outweighed considerations of other communication interfaces at SICK. The

[19] Actually this does not (re)set the format to be used when sending scan data, but since this is not touched by any part of the software there is no need to reset it. For details see the TDM.

name	bytes	contents	explanation
STX	1	0x02	start byte
ADDRESS	1	0x80	reply address (in our case
LENGTH	2	726	DATA length (not including checksum)
DATA	1	0xB0	CMD
	2		FLAGS
	2		FLAGS = UNIT_{14}^{15}:PSCAN1_{13}^{13}:PSCAN2_{11}^{12}:COUNT_{0}^{10}
	2		SCAN_0 = BLEND_{15}^{15}:DIST_{0}^{12}
	⋮		⋮
	2		SCAN_{360}
	1		padding
CRC	2		CRC checksum from STX to DATA

Table A.9: The SCAN_CONTINUOUS telegram (`0xB0` CMD). The FOO_i^j notation indicates how values are packed into the bits i to j of the DATA. The values are: UNIT = 1 indicating a millimetre scale, PSCAN1 and PSCAN2 = 0 indicating a complete scan, COUNT = 361, BLEND a 1-bit flag indicating whether a reflector has been detected and DIST the distance in millimetre between 0x0000 (0m) and 0x1ff7 (8.183m). Values between 0x1ff8 and 0x1fff indicate errors. Note that this does leave some bits open which are not defined in this mode.

only alternative to get rid of the processor load would therefore be to use an “intelligent” serial I/O board with a separate processor to buffer the data, but for this work this has not been tested because the load has been found to be tolerable.

A.4 mobiled

The mobile platform control is implemented in the `mobiled` program. Internally, the control architecture of the `mobiled` program consists of several parallel threads as in figure A.6. Externally, the `mobiled` program accepts a number of command line arguments listed in table A.10. The meaning of these arguments is as follows:

```
tesche@fermi/pts/2> ./mobiled -h
usage:  mobiled [<option>]
     where <option> is any combination of
     -d :  enable debugging
     -h :  print this help
     -s :  simulate (motors and laser)
     --no-motor / --no-motors :  disable motors
     --no-laser / --no-lasers :  disable lasers
     --no-voltage :  disable voltage graph
```

Table A.10: Command line arguments of `mobiled`.

`-d` enables lots of debugging output. This option is not recommended for normal usage because it can disturb the real-time capabilities of the system due to calls to the `write()` system call by real-time threads.

`-h` prints the arguments as listed in table A.10.

`-s` enters simulation mode. In this mode no hardware is accessed, but only simulated. This mode is useful to run a simulated `robotd`.

`--no-motor / --no-motors` disabled the use of the motors. This mode is only useful for testing the laser scanners.

`--no-laser / --no-lasers` disabled the use of the laser scanners. This mode is useful for testing the odometry.

`--no-voltage` disables the voltage graph. If the `mobiled` does not have access to a graphical console or cannot find the external helper program for viewing `gnuplot` logs use this option to disable the voltage graph. This does not affect the possibility to read out the voltage manually.

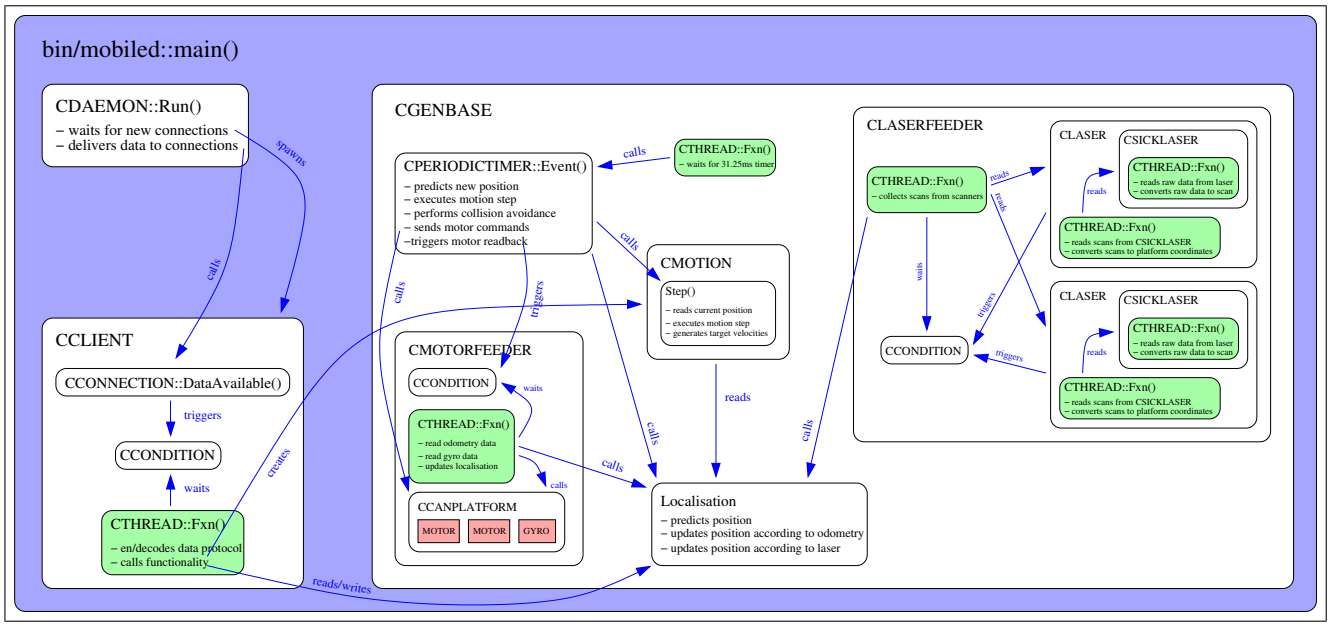


Figure A.6: The control architecture of the server program `mobiled`.

Appendix B

Robot Arm Documentation

This chapter gives an optional, but more verbose documentation of the robot arm used in this work. This documentation consists of a detailed hardware description of the PA10 robot arm as delivered by the vendor in section B.1, details about the custom tool designed for this work in section B.2 and a short introduction into the most important aspects of the robot programming language RCCL used in this work in section B.3.

B.1 The "Mitsubishi PA10" Robot Arm

The robot arm used for the mobile robot system is a 1999 model Mitsubishi Heavy Industries (MHI[1], Japan) "Portable General Purpose Intelligent Arm PA10" in the original japanese version. The PA10 can be ordered in several "levels" from MHI according to table B.1. For this work a level-3 PA10 has been chosen, so the next three subsections will describe the arm, the controller and the *motion control board* (MCB). In each subsection arguments for choosing the PA10 for this work will be pointed out as well as the major problems.

level	scope of devilery
1	arm mechanism only
2	1 + controller
3	2 + motion control board (MCB)
4	3 + operation control section (PC)

Table B.1: Different "levels" of the PA10.

B.1.1 Level 1 – The PA10 Arm

Level 1 of the PA10 consist of only the arm according to the specification data given in table B.2. It weighs only 35 kg, yet it has a rather high payload of 10 kg. This weight is distributed over a kinematically usable length of 950 mm, making it very compact. In comparison, for example the smallest model of Stäubli's RX series, the RX60, weighs 42 kg at a length of 600 mm. A Stäubli RX60 had initially been considered for this work, but thorough considerations soon indicated that 600 mm length are not enough. The auxiliary devices would have to be packed too close and too cramped around the robot so that they would actually hinder the robot when operating them.

[1] `http://www.robot-arm.com`, `http://www.sdia.or.jp/mhikobe-e/products/mechatronic/e_index.html`

weight:	35 kg
payload:	10 kg
motors:	AC servo motors
sensors:	absolute resolvers
number of joints:	7, similar to human arm
kinematical length:	950 mm
positional repeatability:	$\pm$ 0.1 mm

Table B.2: Some PA10 specification data.

The arm has 7 joints arranged in a way to resemble the capabilities (not necessarily the appearance) of a human arm as in figure B.1. It has a *redundant* kinematic, which means that there is generally an infinite number of solutions to the inverse kinematics. This also means that parts of the arm can move without changing the pose at the end effector. Even though this feature is not used in this work, it can be useful in avoiding collisions in narrow environments, where these *self motions* can be used to avoid obstacles while still driving a desired motion. It can also be used to avoid some of the problems associated with singularities[2].

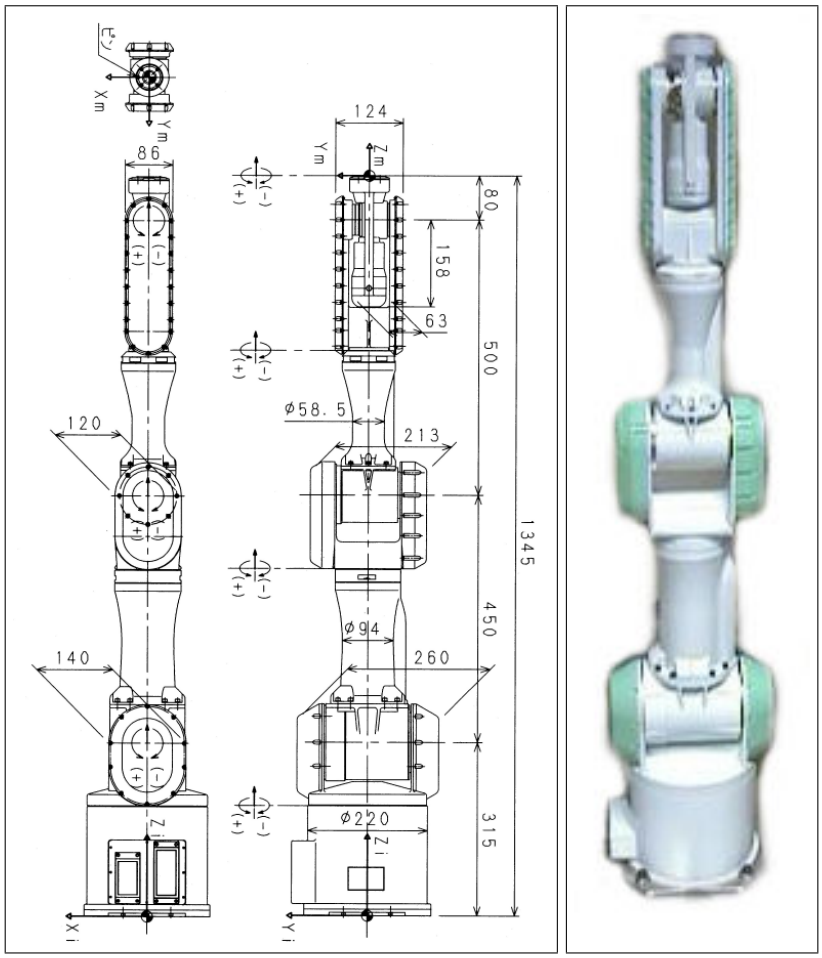


Figure B.1: The PA10 arm with its 7 joints. All lengths are in millimetres.

[2] See [Scherer 1998].

The joints are driven by brushless AC motors over a gear box and position resolvers, all integrated in the compact casing. Contrary to a RX60 the resolvers on the PA10 can report the absolute joint position without needing calibration[3]. The exact specifications of the joints are given in table B.3.

This table shows one interesting point highlighted in gray: The last joint has no mechanical, but only a logical limit. While for some operations this may be a useful feature it raises a problem with the position resolver. The operating manual says (literally)[4]:

> *If power is re-supplied when lost at a position exceeding ±180 degrees, initial angle is not recovered, but an angle displaced by 180 degrees is given.*

Practically, if at a position of more than 180 degrees the angle is displaced by 180 degrees, this means that the arm may come up with 360 degrees error in the zero position of the last joint. This is a severe problem because cables leading to the tool may/will get damaged if the arm is moved when being in this condition.

joint	type	mech limit [°]	servo limit [°]	soft limit [°]	vel limit [rad/s]
1	rotation	±180	±178	±177	±1
2	swing	± 94	± 92	± 91	±1
3	rotation	±180	±175	±174	±1
4	swing	±143	±138	±137	±2
5	rotation	±270	±256	±255	±2Π
6	swing	±180	±166	±165	±2Π
7	rotation	±∞	±361	±360	±2Π

Table B.3: Specifications of the original PA10 joint parameters.

B.1.2 Level 2 – The PA10 Controller

A level 1 PA10 is only useful for people wishing to design their own controller. For all others the level 2 PA10 comes with a MHI controller. The main specifications of this controller are given in table B.4.

The controller is very compact. The logic and power boards inside it use only a fraction of the available space in the box – most of the space is used by the power supply which transforms the japanese standard 100 VAC down to 100 VDC for the power amplifiers. The small size and the fact that it is possible to run the power amplifiers with, for example, only the 48 VDC present on the mobile platform makes it possible to integrate the controller into the mobile platforms without larger modifications.

Internally this controller has several microprocessors – one for running a control loop for each joint and one for global tasks and communication over the ARCNet interface. ARCNet is

[3] More exactly, the RX60 can also do this for some time (up to 2-3 weeks) without power because its incremental encoders are backed up with an auxiliary battery and keep counting even when the main power is switched off. Once this battery has run flat the RX60 “forgets” the position and has to be calibrated.

[4] See [Mitsubishi PA10 Operating Manual].

physical dimensions:	225 × 343 × 395 mm (W×H×D)
weight:	14 kg
external operating voltage:	100 VAC, 50/60 Hz, 1 kVA
internal operating voltage:	100 VDC
external interface:	RJ11 socket, HYC2485S modulated twisted pair 5 Mbps ARCNet
joint controller cycle time:	10 ms
safety timeout watchdog:	300 ms (default, can be changed)

Table B.4: Specifications of the PA10 controller.

a token ring network where stations are only allowed to send data when they are given the token by the network arbiter. In addition to this, the protocol is designed so that the controller will never send packets on its own if not asked to do so by the host. Due to this mechanism there are no collisions on the bus and real-time behaviour can be guaranteed. This real-time behaviour is needed because the joint controllers have to be given new command data at a constant rate of 10 ms. Failure to do so will result in jerky motions and may damage the arm and the objects it manipulates.

The joint controllers in the PA10 provide two different type of motor commands:

1. Velocity commands and
2. torque commands.

Torque commands can be used to access the motors at a very low level when the goal is to obtain full control and responsibility and to implement a motion control completely separate of what the existing higher levels provide. They can also be useful when implementing force control, in particular because it is possible to have only some of the seven joints in torque mode. For this work they are not used because implementing a motion control at a cycle rate of 10 ms using torque commands involves a lot of difficulties, one of which being that it is very tricky to keep the arm steady because due to gravity a torque of zero does not mean that the arm will be standing still. This work therefore uses only the velocity command mode.

One thing that the PA10 controller does not offer is a *position command* mode, where the joint controller accepts real positions and not velocities. Such position commands of course have to be ensured to be reachable in one cycle given the mass, velocity and limited acceleration of the arm, but are very useful if properly implemented. The point is that without having position commands some higher level controller has to be implemented for bringing the arm to a target position, and if that controller runs too slow it may perform badly.

B.1.3 Level 3 – The PA10 Motion Control Board / PA-Library

A level 3 PA10 consists of the arm and the controller as described above plus the “motion control board” (MCB). The MCB is an ISA board (see figure B.2) to be plugged into the computer that is to become the controlling host for the PA10 (newer versions use a PCI board). It has a single ARCNet interface to connect to a single robot. Apart from this interface it also has a complete computer with its own CPU, memory and operating system, independent from the host CPU and operating system. It is intended to be used by all those customers who want a high level access to the arm.

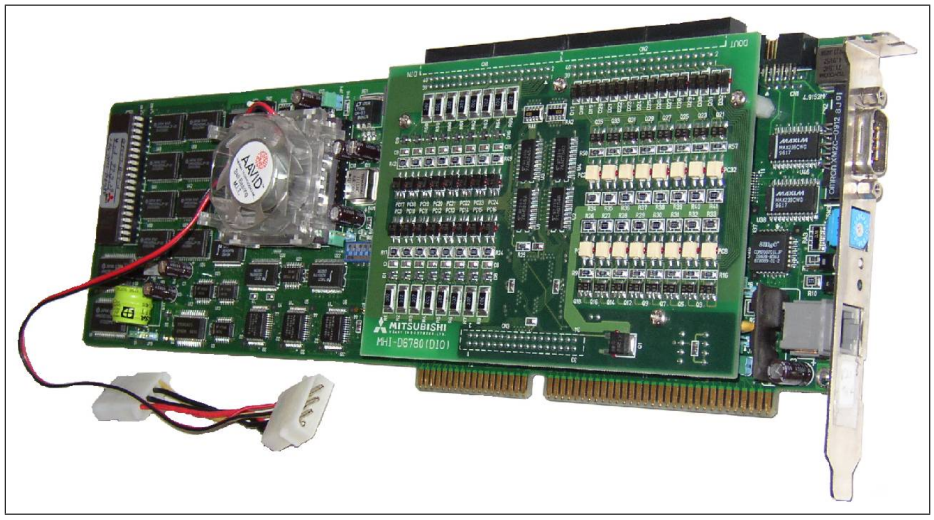


Figure B.2: The Motion Control Board (MCB).

The MCB can perform all the computations needed for moving the arm including computing a trajectory to a cartesian target, generating setpoints along that trajectory, applying some kind of inverse kinematics to obtain joint level setpoint and sending them to the arm via the ARCNet interface. According to the manual the trajectory generator uses the *resolved motion rate control* (RMRC) approach to generate the motions[5].

The fact that the programming manual mentions the term “RMRC” (resolved motion rate control) suggests that trajectory generation is actually done by applying the inverse of the arm jacobian, or, since the arm has a redundant kinematic, rather the pseudo-inverse of the jacobian[6]. RMRC translates cartesian *velocities* into joint *velocities*, which is consistent with the fact that at the lower ARCNet level the controller accepts velocity commands. In order to move the arm to a *position* the MCB has to apply RMRC in a closed loop even if the arm is no longer to be moved, treating the difference in position as a virtual velocity.

No official documentation about the MCB itself exists. The only way the MCB can be used is by purchasing the separately available “PA-Library” (PA-Lib). Since this library comes with source code it can be seen that it implements no functionality on its own, but only passes its function parameters to the MCB. It is therefore limited to MCB hardware features and does not allow any user enhancements. Its key features are briefly described in section 4.2.1 and compared to the *robot control c-library* (RCCL) used in this work.

B.1.4 Newer PA10 Versions

In 2002 MHI has revised the PA10 design and henceforth only sells the PA10-6C with 6 and the PA10-7C with 7 joints, an excerpt of whose specifications is given in table B.5. They differ in a few details from the original PA10, for example in that they support the german operating voltage of 240 VAC and comply to european safety regulations. Other differences include the fact that they have built-in pneumatic tubes and electric cables running from the base to the lower arm to be used freely by the customer.

[5] See [Whitney 1969] and [Whithey 1972].
[6] See [Scherer 1998] and section B.3.4 in the appendices.

	PA10-6C	PA10-7C
number of joints:	6	7
weight:	38 kg	40 kg
payload:	10 kg	
kinematical length:	930 mm	

Table B.5: Some specifications of the newer PA10-6C and PA10-7C.

The most important difference is that they use a different kinematic. The PA10-6C appears to differ in that it has only 6 joints and looks different (see figure B.3), but this is not a kinematical difference. All of its joint locations, zero points and scales coincide with the joints of the PA10-7C so that – except from the non-existent joint E1 that has to be assumed to have a constant value of zero – the same kinematic can be used. The real difference is that they are both shorter (930 mm) than the old version (950 mm), meaning that the kinematic routines must be aware of this and that not all manipulations of the old version in a given and fixed scenario are guaranteed to succeed on the new version.

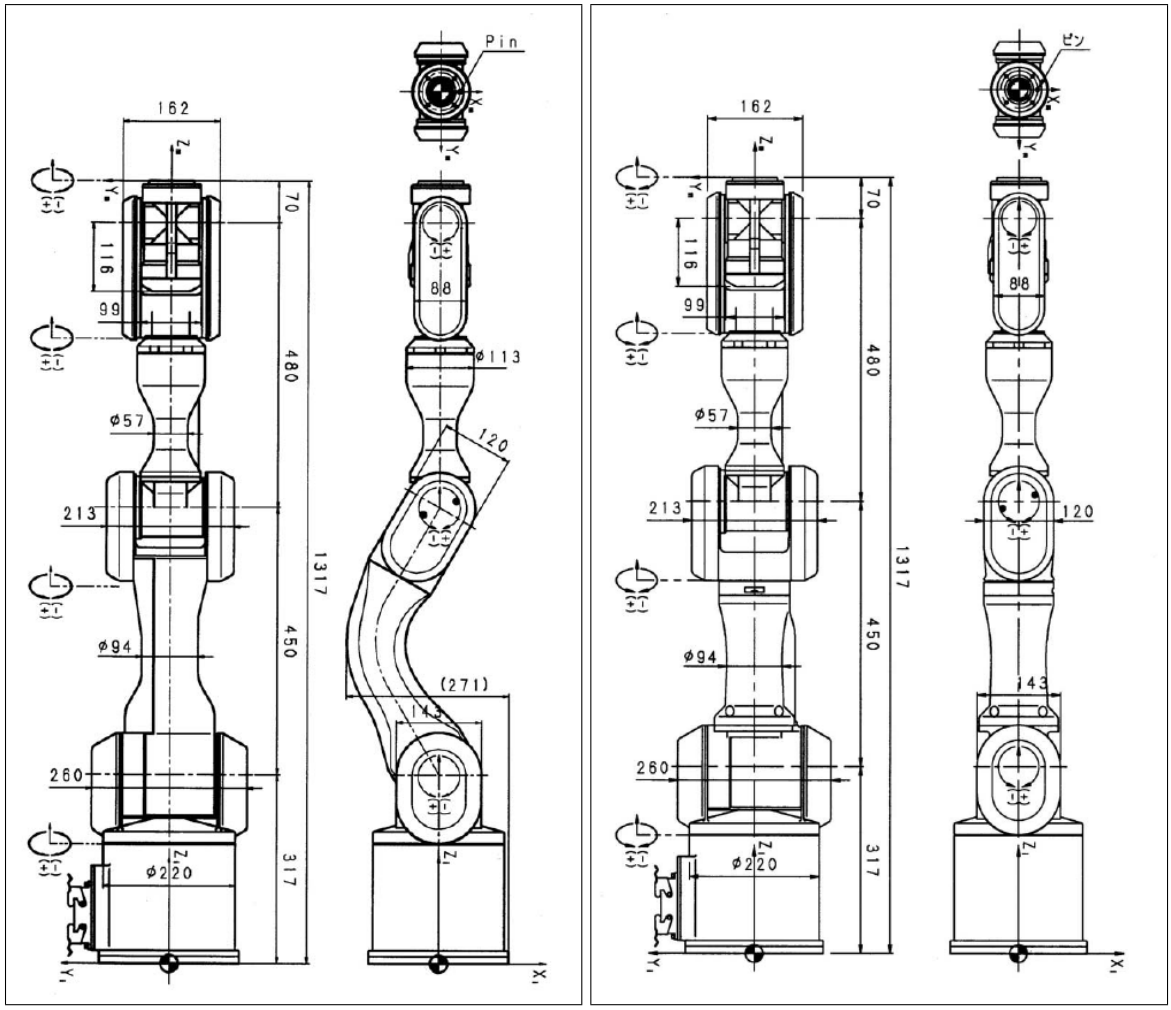


Figure B.3: The PA10-6C (left) and PA10-7C (right). All lengths are in millimetres.

Other differences exist in the controller, which now has an optical ARCNet interface requiring a different ARCNet board to access it, and the MCB, which now comes as a PCI board. The protocol spoken over the ARCNet has changed – some commands take additional arguments and the endianess has changed – making the new versions incompatible at the low level.

B.2 The Robot Tool

The purpose of a robot arm is to interact with its environment, which in the context of this work means to grasp several types of tubes, carry them around, feed them to devices and operate these devices. In order to do that, the robot needs a proper *tool* equipped with (at least) an actuator and (perhaps multiple) sensors. This tool is generally not readily shipped with the robot simply because the robot manufacturer cannot tell what type of tool the customer may be needing. Instead, the robot arms come with flanges that allow the customer to mount – within obvious limitations – whatever tool he wants.

In industry robots are mostly used for a single and rather simple repetitive task requiring only very limited, but sometimes also very specialised tool capabilities. As long as the task is not changed the tool usually does not have to be changed too. The problems start when the task gets too complex, for example when not only one type of object has to be manipulated. In these cases a single tool that fits all requirements can often not be built and a *tool changer* has to be used.

A tool changer allows the arm to change the tool autonomously, eliminating the need to build a complex and large tool for all purposes in the first place. This flexibility, however, does not come without a price, which is that the changing mechanism itself is usually quite big. For industry applications using large robots this may not be a problem, but for the service robot in this work it is.

One of the greatest problems with planning motions for a service robot is to ensure that it has maximum manoeverability **and** keeps the maximum possible safety distance from objects in the environment. Given that the operational range of a small robot like the PA10 is rather limited and most of that operational range is already needed to keep the mobile platform at a safety distance this is not a trivial task, and a big tool changer and the end of the arm would complicate this even more. As a result, a tool changer has to be considered unsuitable for such a robot and was not used in this work.

Instead, a single customised and very compact tool has been designed and fixedly mounted to the PA10. All sensors (notably the camera) and actors (the gripper fingers) are placed as close together as possible to not waste valuable operational range for offset movements between them. The resulting tool can be seen in figure B.4. The force/torque sensor, the electric gripper and the gripper fingers will be described in the following subsections, and the camera is described in chapter 5.

B.2.1 The “ATI Force Torque Sensor”

One of the tool sensors is an “ATI Industrial Automation[7] (ATI, USA) Gamma SI-65-5” *force/torque sensor* (FTS) with 6 *degrees of freedom* (DOF). The FTS is mounted at the wrist as

[7] `http://www.ati-ia.com`. German distributor: Fa. Schunk. `http://www.schunk.de`

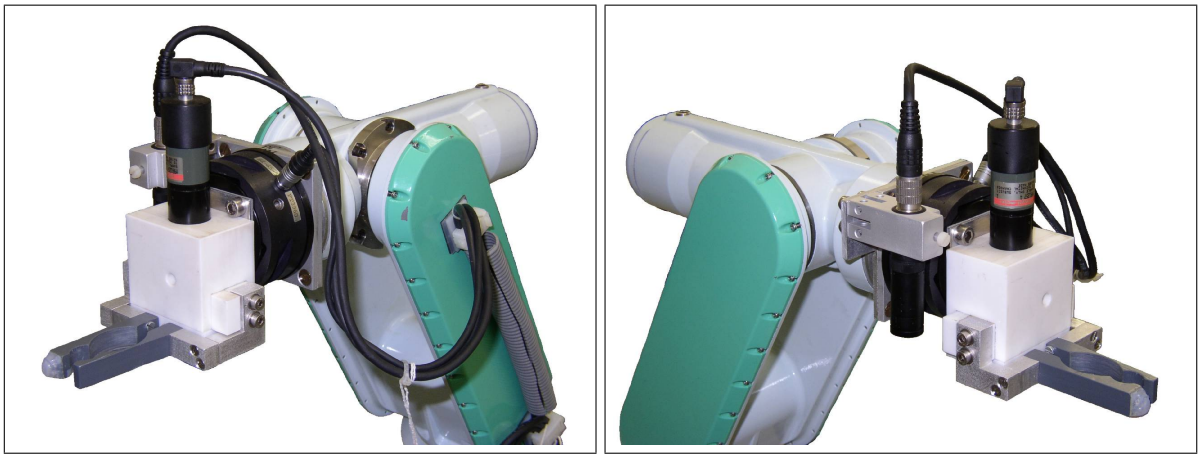


Figure B.4: The tool with the camera, force/torque sensor and gripper.

in figure B.5 to measure forces (3 DOFs) and torques (3 DOFs) along/about all three cartesian axes applied to or by the gripper. It is the only tactile sensor the arm has got, i.e. there are no explicit force sensors in the grippers.

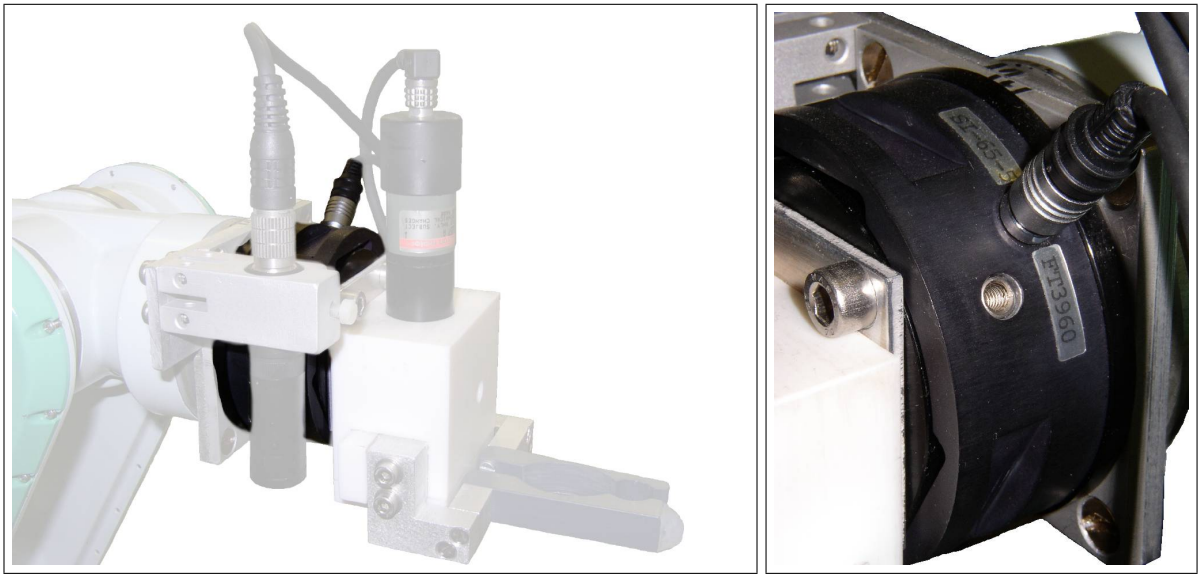


Figure B.5: The wrist mounted force/torque sensor (FTS).

The basic principle of operation of such a FTS is Hook's Law (the spring law), except that no ordinary "spring" is used. The back-end of the FTS that is mounted to the arm and the front-end where the gripper is mounted to are not totally fixed, but connected only by an elastically deformable object. On this object strain gauges are attached to measure the slight deformations that occurs when forces are applied to it. Using a proper placement of the strain gauges forces and torques along and about all three cartesian axes can then be computed.

The biggest problem with such a construction is that not only external forces can exert deformations, but also thermic expansion of the sensor. In theory this effect can be compensated by using two properly placed strain gauges in a bridge, but since in practise the thermic expansion

is not always uniform an error remains[8]. Summing up this error, the disturbances caught on the analog cable and the sampling errors in digitisation the overall error with the Gamma SI-65-5 is about ± 0.2 N (for the forces). This is a bit troublesome because the range of force values in which force control is usually used in this work is only about ± 2 N, so the error is about 10%. It means that it is not possible to adjust forces to an infinite accuracy.

The FTS is connected to the PC by an ISA bus *digital signal processor* (DSP) receiver board that does all the necessary digitisation and conversion from raw strain gauge values into calibrated force/torque values, as well as low-pass filtering to eliminate noise. The DSP board is integrated into the controlling PC by a linux device driver realized as a kernel module. This device does all the necessary initialisation and allows `ioctl()` calls to read out the forces and torques, readily converted into N and Nm.

On a higher level the FTS is transparently integrated into the robot control software by means of a library using the linux device interface. The user of the robot control software can read out the FTS values from a data structure that is automatically updated with the control frequency of 100 Hz. For a brief description of the force control implemented on top of this see section 4.4.2.

B.2.2 The "phd Electrical Gripper"

The tool's actor is a "phd[9] (USA/Germany) EGP-5MG-8x14-CA-H141" based customised parallel yaw gripper as in figure B.6. It is driven electrically by a conventional DC motor rather than a stepper motor. Therefore it needs a small interface box mounted at the base of the arm containing some electronics to control the opening/closing speed and/or limit the motor current.

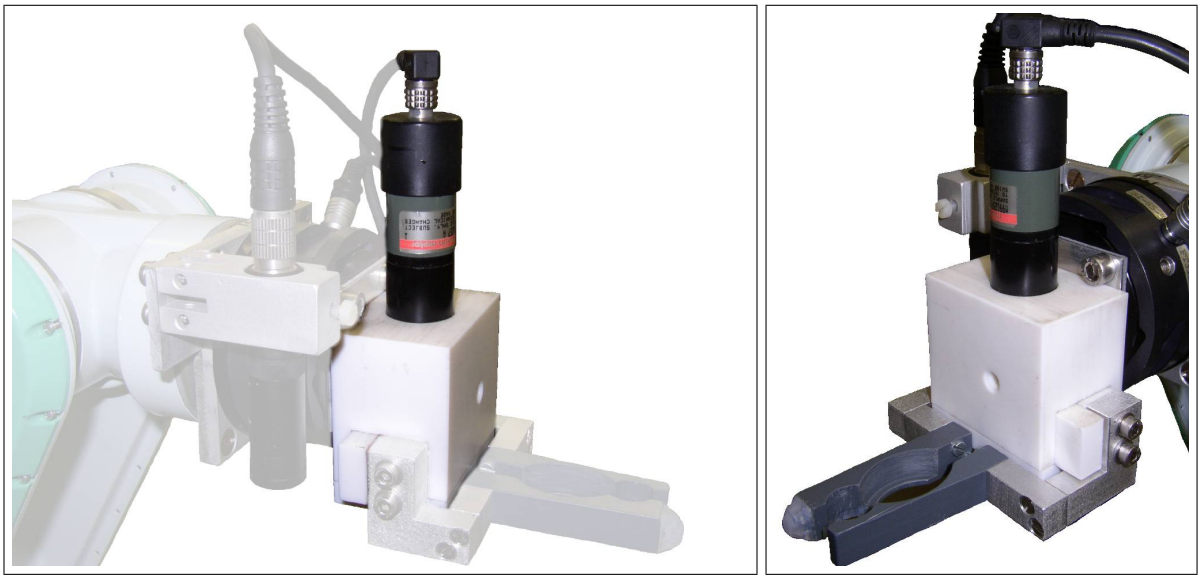


Figure B.6: The electric parallel yaw gripper.

[8]Having an air-conditioning in a sealed laboratory does not necessarily help as the reaction of that air-conditioning to outside weather changes can be quite fast and the resulting temperature distribution in the room quite non-uniform.

[9]Fa. PHD GmbH, Arnold-Sommerfeld-Ring 2, D-52499 Baesweiler, GERMANY.
`http://www.phdinc.com`, `http://www.phd-gmbh.de`

The interface box is connected to two general purpose digital I/O bits on the FTS receiver board. This means that it is accessed by the FTS library via the Linux device driver for this board. The gripper support has been transparently integrated into the robot control software so that the usual commands to open/close it can be used.

The gripper can be commanded to open or close, but neither can an arbitrary position between these two extremes be commanded nor can the actual position be read out. The speed with which it moves can only be influenced indirectly by manually changing the current limit at the interface box. Since the motor is always running this current limit also determines the force with which the gripper holds objects. This may not be an optimal approach for grasping tasks in general, but is fully sufficient for the setup in this work.

One of the biggest problems with grasping is that one has to be sure whether the object in question is properly grasped or not. With no sensors to detect this the only way to ensure a proper grasp is to grasp it carefully. Industry often uses pneumatic grippers because these can apply high forces to obtain a firm grasp. The drawback with these grippers is that their fingers close very quickly, in fact they more snap together rather than close. When the finger clash with the object it may well happen that the object is moved out of position. Since this does applies to most objects in this work a pneumatic gripper can be expected to fail on the required grasping tasks.

An electric gripper like the phd avoids this problem because it closes its finger rather slowly. The fact that electric grippers cannot apply the high forces of pneumatic grippers is not a problem because the objects in this work are quite light-weight and do not at all require high forces to be handled. Quite on the contrary the Cedex tubes are rather fragile, so handling them with limited forces is actually a must.

B.2.3 Gripper Fingers

The gripper is equipped with two customised fingers as in figure B.7. These fingers have to be designed to be able to safely handle all necessary type of tubes and to operate the necessary devices. Operation of devices means “opening of lids” and “pressing of buttons” in the context of this work and is accomplished rather easily with any type finger. In order to open the lids of the centrifuge and fridge the finger basically only has to be able to reach behind it and lift or shift it, and in order to press a button on the membrane keyboard of the centrifuge the finger can be equipped with a piece of silicone rubber. In order to safely handle the tubes more attention is required.

A human operator doing the sample management uses three different tubes for the sample management as shown in figure B.8: A large 50 ml NUNC tube for taking the sample from the reactor (because the centrifuge that is needed later uses them), a smaller 1 ml Cedex tube for feeding the Cedex and a very small 1.5 ml Eppendorf tube for archiving the cell-free supernatant. The initial idea of letting the the robot manipulate all these tubes had to be dropped because no feasible way to manipulate the 1.5 ml Eppendorf tubes could be found. Since the archiving can – although being a waste – also be done in NUNC tubes the tool was designed to be able to grasp only the NUNC and the Cedex tubes.

This restriction also eases another problem, which is that cut-outs have to be milled in the fingers so that they properly align with the round shape of the tubes. The reason for this is that with no cut-outs the contact to the objects would be a two-point contact only, and therefore very

Figure B.7: The customised gripper fingers with curved cut-outs fitting the NUNC and Cedex tubes and a piece of silicone rubber to press buttons on the membrane keyboard of the centrifuge.

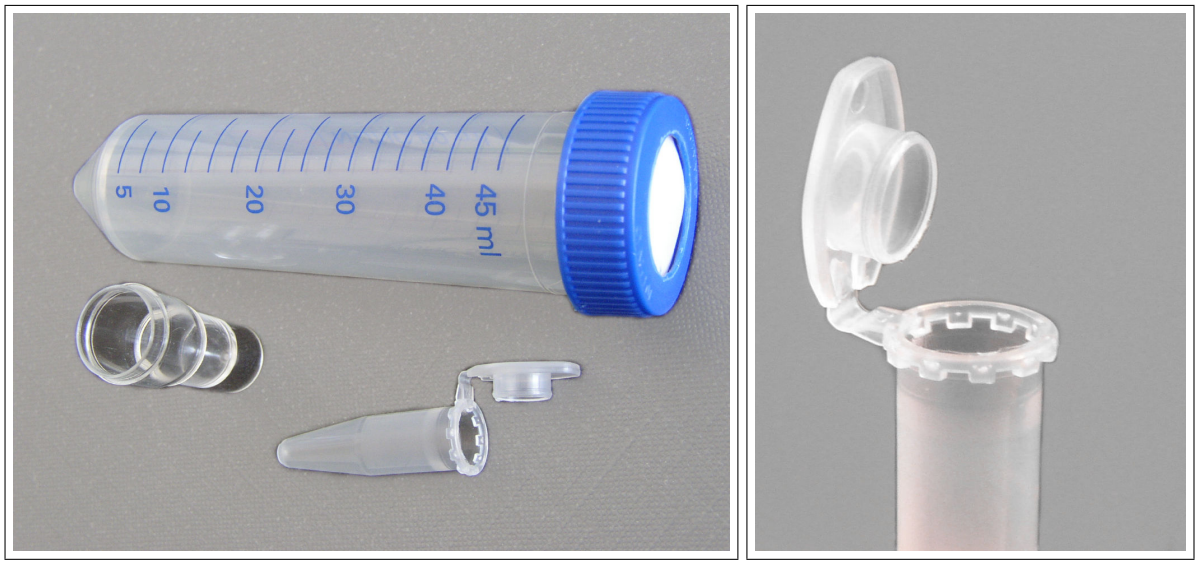


Figure B.8: Three different kind of tubes used for manual sample management. Left image: A 50 ml NUNC tube (upper), a 1 ml Cedex tube (lower left) and a 1.5 ml Eppendorf tube (lower right). Right image: A closeup of the Eppendorf tube.

unstable. With cut-outs, and eventually cut-outs coated with some rubbery material the contact is established plane-on-plane, and therefore much more stable.

Being able to grasp tubes of different diameter means to have cut-outs of different radius, and since the cut-outs cannot be placed into one another this means that the fingers have to be longer. Long fingers in turns raise a problem with the operational range of the arm: If a tube has to be hold *under* some device – as is the case with the pipette – this means that a camera observing the scene cannot be mounted *above* the gripper. It has instead to be mounted with some displacement. If then the tube has to be visually centred above some device this means

that additional translational motions have to be applied to compensate this displacement. These length of these translations correlates with the length of the fingers, and so in order to not waste too much operational range of the arm the fingers have to be kept short. As can be seen in figure B.9 the resulting fingers are quite capable of grasping NUNC and Cedex tubes.

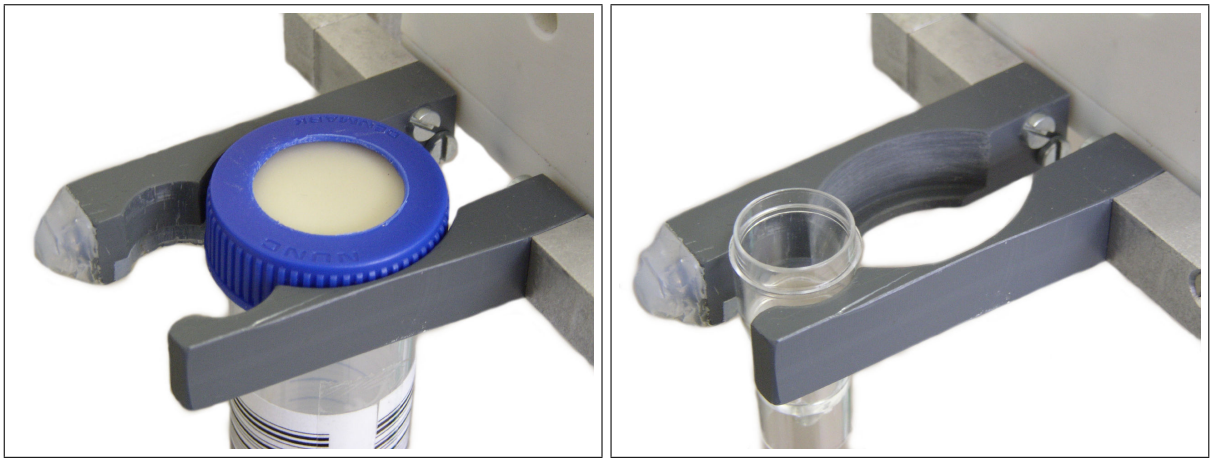


Figure B.9: The gripper fingers holding a 50 ml NUNC tube (left) and a 1 ml Cedex tube (right).

Having only two fingers is the reason why the screw caps of the 50 ml NUNC tubes cannot be used, leading to an important modification. The caps have to be drilled and a special membrane has to be placed into them for sealing the tube. This membrane does not tear if pinched with a needle, but instead closes again perfect enough to let no fluid drop out of the tube afterwards if the tube is not jogged too much With these modified NUNC tubes all biotechnical requirements are met[10] and the robot can automate the sample management.

B.3 RCCL

In this section a short introduction into the most prominent features of *Robot Control C-Library* (RCCL) by John E. Lloyd and Vincent Hayward[11] will be given. This introduction is not meant to be a user's guide to RCCL, but only to motivate the decision to use RCCL in this work. This decision is based on the fact that RCCL has support for many extremely useful features that commercial robot vendor's proprietary programming languages lack.

B.3.1 Mathematic Computations

Controlling robots involves a lot of computations about joint and/or cartesian positions. RCCL offers a lot of useful data types as well as the most commonly used routines to work with them. These include, but are not limited to:

B.3.1.1 Transforms

The most prominent data structure is the *homogenous transformation* – short: *transformation* or *transform* – called `TRSF` in RCCL. A transform defines a coordinate system (frame) and can

[10] See [Poggendorf 2004].
[11] See [Lloyd and Hayward 1989].

be regarded either as describing an absolute *pose* (position and orientation) in 3d-space or as a relative change of pose (motion) with respect to a reference frame[12]. In case the reference frame is the coordinate system origin these two meanings coincide. A (homogenous) transform T is a 4×4 (homogenous) matrix

$$T = \begin{bmatrix} & R & & \vec{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$R = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}$$

is a 3×3 rotational matrix consisting of three unit vectors $\vec{n}$, $\vec{o}$ and $\vec{a}$ describing the x, y and z-axes of the new coordinate system as seen in the old coordinate system and

$$\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

is the position vector. Transforms can be easily inverted (meaning seeing the old coordinate system from the new one, or a motion backwards) because the inverse R^{-1} of a rotational matrix R is simply its transpose R^T.

$$T^{-1} = \begin{bmatrix} & R^T & & -R^T\vec{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transforms can be multiplied like any other matrices like in

$$T_{\mathrm{BASE}} \cdot T_6 \cdot T_{\mathrm{TOOL}}$$

to describe a combined motion or a combined pose that is concatenated from several relative motions. Standard rules for matrix multiplication apply if a chain like this is to be solved for one of its components.

RCCL offers all the necessary functions to do computations with transforms. Unlike that, the PA-Library does have a comparable data structure `MATRIX` but only uses it as a target definition in the command `pa_mov_mat()`. It does not provide any routine to do computations with these matrices.

B.3.1.2 Kinematic

Another important aspect of robots is to have access to their kinematics, that is, to compute a cartesian position from a set of joint angles or backwards.

Given that the inverse kinematics has multiple solutions for a 6 DOF robot it is necessary to manually choose a solution. In addition to this some joints may be able to move more than

[12] See [Fu, Gonzales and Lee 1987].

360 degrees, making it necessary to supply *reference angles*. If no reference angles are given these joints may be driven close to a limit on one side during sequences of motions where the same position with an angle $\pm$ 360 degrees may be a lot more relaxed. In addition to that robot motions are usually given as targets only, even if the user had an explicit trajectory in mind. In case of a joint close to its limit the *trajectory generator* may think it necessary to drive a joint in precisely the opposite direction as the user intended, even though the final position will be correct[13]. All this makes it necessary to have a possibility to intervene with the kinematic to adjust things to how the user wants them to be.

In case of a redundant robot as the PA10 the inverse kinematic actually has an infinite number of solutions and even more complex schemes to choose one are required. One possible alternative is to take that solution where the joints are as far away from their limit as possible, only that now this is a question of minimising a continuous error function rather than selecting one of a discrete set of solutions. Again, if no attempt is made to keep the joints away from their limits problems with motions will occur.

RCCL provides the routines `fwdKinematics()` and `invKinematics()` to do computations with the kinematics, accepting a *configuration bitmask* as well as reference angles to allow the user to have some control over the selection of the solution. It does not implement a selection scheme or a trajectory generator for redundant robots, although these could be implemented by means of control functions by the user. The PA-Library on the contrary does not offer any access to kinematic computations.

B.3.2 Position Equations

Another very useful construct are position equations. A position equation is a highly flexible way of defining motion targets. For simple programs with just a few static targets it may be feasible to pre-compute all target positions, but in general that has to be considered a bad idea.

If a program has a lot of motion targets this means that all the affected absolute positions have to be computed in advance, which may be a bit of work. If in addition to that one of the targets is dynamic (that is, if it changes between two motions) this means that the affected absolute positions have to be re-computed each time this target is accessed.

As an example suppose a table with a tube storage rack on it as in figureB.10. In order to grasp tubes from the racks their absolute positions have to known. If the table or the rack is moved this means that these positions have to be re-computed. Of course this is already greatly simplified if it is possible to do computations with transform, but still it requires a lot of computations. An even more convenient solution is therefore to use position equations and let the robot control software do all this alone.

Position equations are based on the fact that transforms are matrices and can be concatenated as in

$$T_{\mathrm{TABLE}} \cdot T_{\mathrm{RACK}} \cdot T_{\mathrm{TUBE},i}$$

[13]The easiest and most disturbing example of such a situation is screwing a screw into a screw thread. The user has a very explicit idea of which was to rotate the joint in question, but when only the target orientation is given it cannot be guaranteed that the trajectory generator will agree with that. Using a joint space motion of that joint is of course one way to ensure that the screwing will work, but is unlikely to be applicable because an overlaying cartesian motion is needed to compensate for the thread pitch and no robot control software supports mixing joint and cartesian motions for individual joints at the same time.

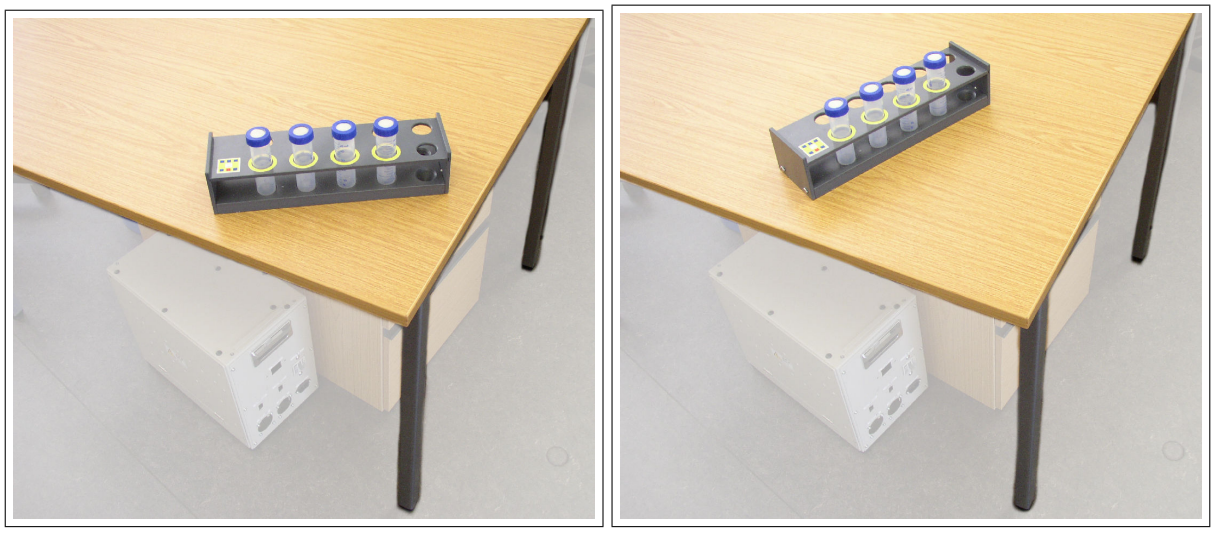


Figure B.10: A table with a tube storage rack on it. Between the left and the right image the rack has been moved, requiring to re-compute the absolute position of all tubes if they are to be grasped.

to describe a more complex target frame, in this case the absolute position of tube i. An illustration of this chain of transforms can be seen in figure B.11. In a position equation two different ways to reach a target frame are given, one of which must contain the robot transform $T_6(t)$ and the other not. Taking the example with the table and wanting to grasp tube i the position equation

$$T_{\text{BASE}} \cdot T_6(t) \cdot T_{\text{TOOL}} \quad = \quad T_{\text{TABLE}} \cdot T_{\text{RACK}} \cdot T_{\text{TUBE},i}$$

is most likely to be *false* at the beginning of a motion – the two sides are not equal. What the trajectory generator does is to introduce an additional transform $T_{\text{DRIVE}}(t)$ that catches the difference. So the real (internal) position equation is

$$T_{\text{BASE}} \cdot T_6(t) \cdot T_{\text{TOOL}} \quad = \quad T_{\text{TABLE}} \cdot T_{\text{RACK}} \cdot T_{\text{TUBE},i} \cdot T_{\text{DRIVE}}(t) \tag{B.1}$$

which can always be *made* true. This equation is then solved for the initial $T_{\text{DRIVE}}(0)$.

$$T_{\text{DRIVE}}(0) \quad = \quad T_{\text{TUBE},i}^{-1} \cdot T_{\text{RACK}}^{-1} \cdot T_{\text{TABLE}}^{-1} \cdot T_{\text{BASE}} \cdot T_6(0) \cdot T_{\text{TOOL}}$$

If we assume the motion to have a total duration of t_E at the end of the motion

$$T_{\text{DRIVE}}(t_E) \quad = \quad \mathcal{I}$$

must hold because the initial position equation has to become true – at the target both sides of the equation are equal. All that the trajectory generator now does is to bring $T_{\text{DRIVE}}(t)$ to $\mathcal{I}$ in the computed motion time t_E by scaling the translation and rotation. By inserting this $T_{\text{DRIVE}}(t)$ into equation B.1 and solving it for

$$T_6(t) \quad = \quad T_{\text{BASE}}^{-1} \cdot T_{\text{TABLE}} \cdot T_{\text{RACK}} \cdot T_{\text{TUBE},i} \cdot T_{\text{DRIVE}}(t) \cdot T_{\text{TOOL}}^{-1}$$

the robot is then brought to the target without that the user ever had to explicitly compute values for T_6.

If now the table is moved all that has to be done is the new value of T_{TABLE} has to be measured. No other transforms have to be changed, in particular none of the $T_{\text{TUBE},i}$. All

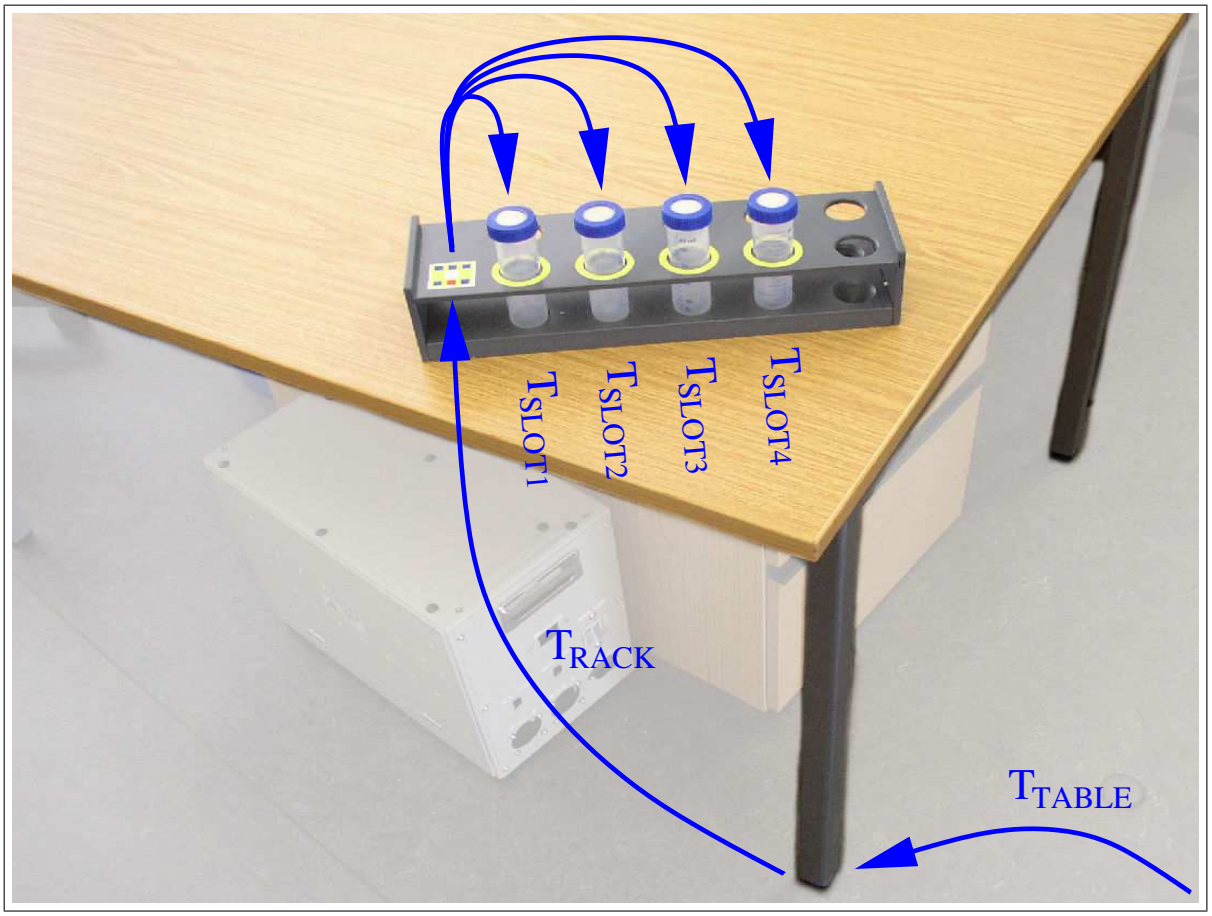


Figure B.11: The same situation as in figure B.10, only now relative transforms are defined to link the objects. If now the rack is moved only T_{RACK} has to be changed and the rest can be computed automatically by the robot control software.

the motion equations can remain unchanged because RCCL will internally compute everything that is necessary to do a motion to the changed target. This greatly simplifies things when multiple targets or multiple transforms per target are involved, and having a position description split into multiple transforms is always a good idea because it makes a complex position more understandable.

RCCL offers all the necessary functions to create and handle position equations and solve them for any necessary transform (or sub-chain of transforms). The PA-Library does not offer anything comparable to a position equation.

B.3.3 Arbitrary Motions

So far the positions equations from section B.3.2 contained only constant transforms (except $T_6(t)$), however, having one or more *variable* is an interesting extension. Using variable transforms in a position equation means that RCCL cannot compute T_{DRIVE} and therefore the complete motion in advance because it does not know about the changes that will occur.

What it does is it computes T_{DRIVE} from the *initial* values of the variable transforms and starts the motion as normal. If the variable transforms are not actually changed this leads to the same motion as with constant transforms. If they are changed, RCCL can do nothing but

apply all the change ontop of the normal motion immediately as in figure B.12. It cannot blend these changes into the motion as this would require re-computing all the motions parameters including t_E, which is not supported.

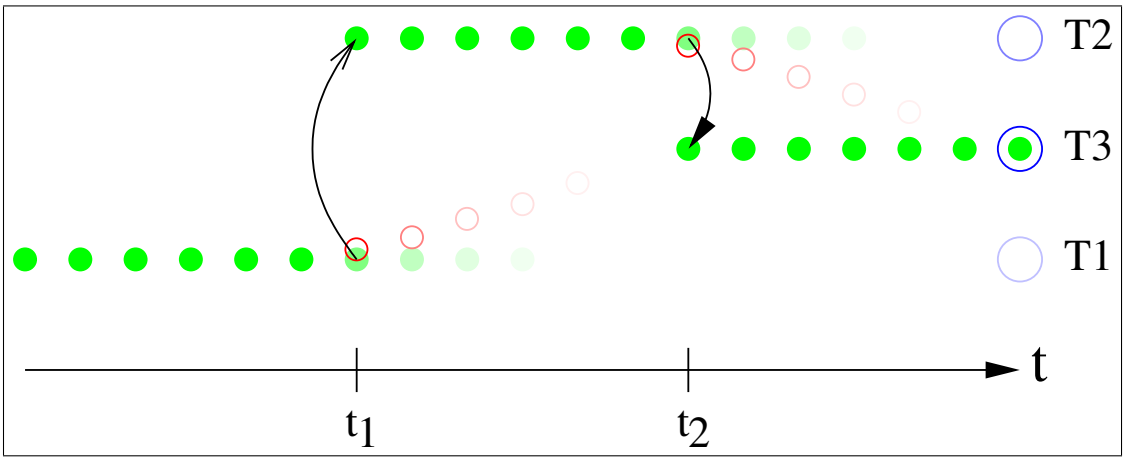


Figure B.12: Setpoint generation for moving targets: At $t = t_1$ the target was moved from $T1$ to $T2$ and at $t = t_2$ it was moved again to $T3$. The current trajectory (the fading green dots) is left and the change applied in full immediately and not blended smoothly (the fading red circles).

Monitor functions can be attached to a transform to be called in every cycle of the trajectory generator, which means that the transform can be modified in real-time. This way a normal motion may be superimposed by an arbitrary offset as long as the offset does not violate the robot's acceleration and speed limits.

A special case occurs if the constant part of the motion is actually `NULL` because the robot is already at the target position of the motion. Normally such a motion would take no time and complete immediately, but with variable transforms it is possible to manually set the motion time t_E to infinity (or any other applicable time). During execution of such a motion the robot does no pre-planned moves but only react to changes of the variable transforms. This way a truly arbitrary motion may be achieved, again, as long as the offset does not violate the robot's acceleration and speed limits.

Applications of arbitrary motions are for example spline curves as an extension of the trajectory generator and force control, where correctional movements obtained by measurements from a FTS are superimposed onto an otherwise constant motion to prevent excess forces from damaging the robot or other equipment (for details see section 4.4.2).

RCCL's position equations and variable transforms offer all the support that is necessary to implement these arbitrary motions. The PA-Library has limited capabilities to modify a motion (see `pa_set_mat()`, `pa_set_wav()` and `pa_odr_xyz()`), but these changes are still smoothed by the trajectory generator and the approach is therefore not suitable for real force control. It is any any case not possible to do a fully arbitrary motions with them.

B.3.4 Singularities

One interesting application of a low-level intervention during motions is *singularity robustness*. Singularities are situations (sets of joint angles $\vec{\theta}$) where the robot's jacobian matrix $J(\vec{\theta})$ becomes singular. The jacobian represents the partial derivatives of each cartesian DOF p_i according to the robot's kinematic $f_{\mathrm{KYN}}(\vec{\theta})$ with respect to each joint DOF θ_j as

$$J(\vec{\theta}) \;=\; \left[\frac{\partial p_i}{\partial \theta_j}\right]$$

It is the first order (linear) term of the *taylor series expansion*

$$f(x) \;=\; \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

of the robot's kinematic $f_{\mathrm{KYN}}(\vec{\theta})$ around some $\vec{\theta}_0$. For example, for a robot with 6 DOFs and 6 joints like the PUMA type robots or the PA10-6C the jacobian is given by[14]

$$J(\vec{\theta}) \;=\; \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \frac{\partial p_x}{\partial \theta_3} & \frac{\partial p_x}{\partial \theta_4} & \frac{\partial p_x}{\partial \theta_5} & \frac{\partial p_x}{\partial \theta_6} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \frac{\partial p_y}{\partial \theta_3} & \frac{\partial p_y}{\partial \theta_4} & \frac{\partial p_y}{\partial \theta_5} & \frac{\partial p_y}{\partial \theta_6} \\ \frac{\partial p_z}{\partial \theta_1} & \frac{\partial p_z}{\partial \theta_2} & \frac{\partial p_z}{\partial \theta_3} & \frac{\partial p_z}{\partial \theta_4} & \frac{\partial p_z}{\partial \theta_5} & \frac{\partial p_z}{\partial \theta_6} \\ [R_1]_{3,2} & [R_2]_{3,2} & [R_3]_{3,2} & [R_4]_{3,2} & [R_5]_{3,2} & [R_6]_{3,2} \\ [R_1]_{1,3} & [R_2]_{1,3} & [R_3]_{1,3} & [R_4]_{1,3} & [R_5]_{1,3} & [R_6]_{1,3} \\ [R_1]_{2,1} & [R_2]_{2,1} & [R_3]_{2,1} & [R_4]_{2,1} & [R_5]_{2,1} & [R_6]_{2,1} \end{bmatrix}$$

where

$$\begin{aligned} R &= \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} && \text{(the rotational component of the robot's transform)} \\ R_j &= \left[\frac{\partial R}{\partial \theta_j} R^T\right] && \text{(as abbreviation)} \\ [M]_{r,c} &= M_{r,c} && \text{(matrix element in row number } r \text{ and column number } c\text{)} \end{aligned}$$

This jacobian J can be used to compute small differential changes $\Delta\vec{p}$ in the cartesian pose from small incremental changes $\Delta\vec{\theta}$ in the joint angles via

$$\Delta\vec{p} \;=\; J(\vec{\theta}) \cdot \Delta\vec{\theta}$$

The RMRC method for trajectory generation implemented on the MCB inverts this equation to compute small incremental changes $\Delta\vec{\theta}$ in the joint angles from small differential changes $\Delta\vec{p}$ in the cartesian pose via

$$\Delta\vec{\theta} \;=\; J^{-1}(\vec{\theta}) \cdot \Delta\vec{p}$$

For redundant robots like the PA10 where no closed solution to the inverse kinematics exists this is actually the only way to generate motions. The only drawback is that in this case J

[14] See [Scherer 1998].

is not square and the normal inverse cannot be used, but this can be overcome by using the pseudo-inverse.

If J becomes singular J^{-1} does not exist, with the practical upshot that in computing it a *division by zero* occurs and the computed $\Delta\vec{\theta}$ are therefore $\pm\infty$ for one or more joints. If the program does not abort immediately because of SIGFPE[15] it means that it will abort shortly afterwards because the robot cannot do motions with a higher velocity than its physical limit.

Numerous solutions to this problem exist[16], but they all involve slowing down along the trajectory or slightly deviating from it. These can quite easily be implemented with arbitrary motions in RCCL. Since the PA-Library does not allow arbitrary motions it cannot offer any means to deal with singularities.

B.3.5 (Virtual) Robot Cooperation

As a final aspect the cooperation of two or more robots has to be mentioned. Since this work uses only one robot this appears to be a useless idea at first sight, however, this will be shown to be not true.

B.3.5.1 Cooperation

Robot cooperation is an important feature when handling large or complex objects – a single robot may physically not be able to lift the object because it is either too heavy or because it cannot grasp it near its centre of gravity (in which case it will drop out of the gripper if lifted). With two robots the same task may be physically possible, but is more complex to program. Basically, to allow two robots to cooperatively carry the object their motions must be synchronised. If they are not synchronised already the slightest positioning deviations can cause excessively high forces to occur inside the object or the robot and – fatally – damage them.

Consider two robots MASTER and SLAVE carrying an object as in figure B.13. The master is intended to perform the real motion and the slave is intended to follow it. In order for the slave to do this it must know where the master currently is, and the easiest way to achieve this is to link its T_6 transform into the position equation like in

$$\begin{aligned} \mathrm{MASTER} &: T_{\mathrm{BASE,MASTER}} \cdot T_6 \cdot T_{\mathrm{TOOL,MASTER}} = T_{\mathrm{TARGET}} \\ \mathrm{SLAVE} &: T_6 \cdot T_{\mathrm{TOOL,SLAVE}} = \mathrm{MASTER} \rightarrow T_6 \cdot T_{\mathrm{OFFSET}} \end{aligned}$$

Note that the slave does not need a base transform (and strictly speaking it also does not need a tool transform). All that has to be done is that T_{OFFSET} has to be properly initialised so that the second equation is valid at the start, then the slave has to be sent into an infinite motion tracking its position equation and then the master is commanded to do its motion. In each cycle RCCL will see that the target for the slave has shifted slightly because $\mathrm{MASTER} \rightarrow T_6$ has changed and will adjust $\mathrm{SLAVE} \rightarrow T_6$ accordingly, thus tracking the motion of the master. Since the master is a real robot with speed and acceleration limits it can in most cases be assumed that the resulting motion of the slave will also be within its speed and acceleration limit, though this cannot generally be guaranteed in advance.

[15] SIGFPE: Floating Point Exception. UNIX signal to indicate numerical problems with the floating point processor.

[16] See for example [Lloyd 1995], [Lloyd and Hayward 1996], [Lloyd 1998] and [Lloyd and Hayward 1998].

Figure B.13: Two robot arms cooperatively carrying an object.

B.3.5.2 Virtual Robots

Whereas it is possible to cooperatively carry an object in the above way it remains awkward because it is the master robot that defines the path. If instead the object is required to move on a specific path the user has to compute in advance what the path of the master robot has to be to result in the desired object motion. Mentioning only one aspect, the kinematic lever between the object and the robot leads to that rotations of the object will translate into translations along a circular arc for the robot. This is a problem because the only motion types that RCCL as well as most other robot programming languages provides are straight line motions in either joint or cartesian space. At first sight it therefore appears as if using a master/slave approach for cooperation is hardly an improvement of the situation.

In order to overcome this problem RCCL implements the notion of *virtual robots*. A virtual robot is a normal robot seen from a cartesian point of view, but has no joint data structures, no kinematic and no hardware interface. It has cartesian velocity and acceleration limits, has a cartesian T_6 position and can be moved in cartesian space according to any position equation. It can not be commanded to operate in joint interpolation mode and none of its joint structures must ever be touched.

With this concept of virtual robots the task of cooperatively carrying an object can now be refined. The idea is to let the object itself be a virtual robot and attach *both* robots to it as slaves with position equations like

$$\begin{array}{lrcl}
\text{LEFT} : & T_6 \cdot T_{\text{TOOL,LEFT}} & = & \text{OBJECT} \rightarrow T_6 \cdot T_{\text{OFFSETLEFT}} \\
\text{OBJECT} : & T_{\text{BASE,OBJECT}} \cdot T_6 \cdot T_{\text{TOOL,OBJECT}} & = & T_{\text{TARGET}} \\
\text{RIGHT} : & T_6 \cdot T_{\text{TOOL,RIGHT}} & = & \text{OBJECT} \rightarrow T_6 \cdot T_{\text{OFFSETRIGHT}}
\end{array}$$

If both robots are set up to passively track the object the same mechanism as above applies: RCCL sees that the object's T_6 changes in each step and corrects each of the robot's T_6 in order to keep the equations true. It translates the motion of the object into motions of the robots, moving them on whatever path is necessary without the user having to do any advance computations at all.

The reason why this idea of virtual robots also helps simplifying a system with just one robot is precisely that complex form of motion the robot may have to follow if it is attached to an object that is moving. If, for example, the robot has to close the lid of the centrifuge this means to do a circular motion. Instead of programming the motion explicitly a virtual robot can be placed so that its z-axis coincides with the hinge of the lid as in figure B.14.

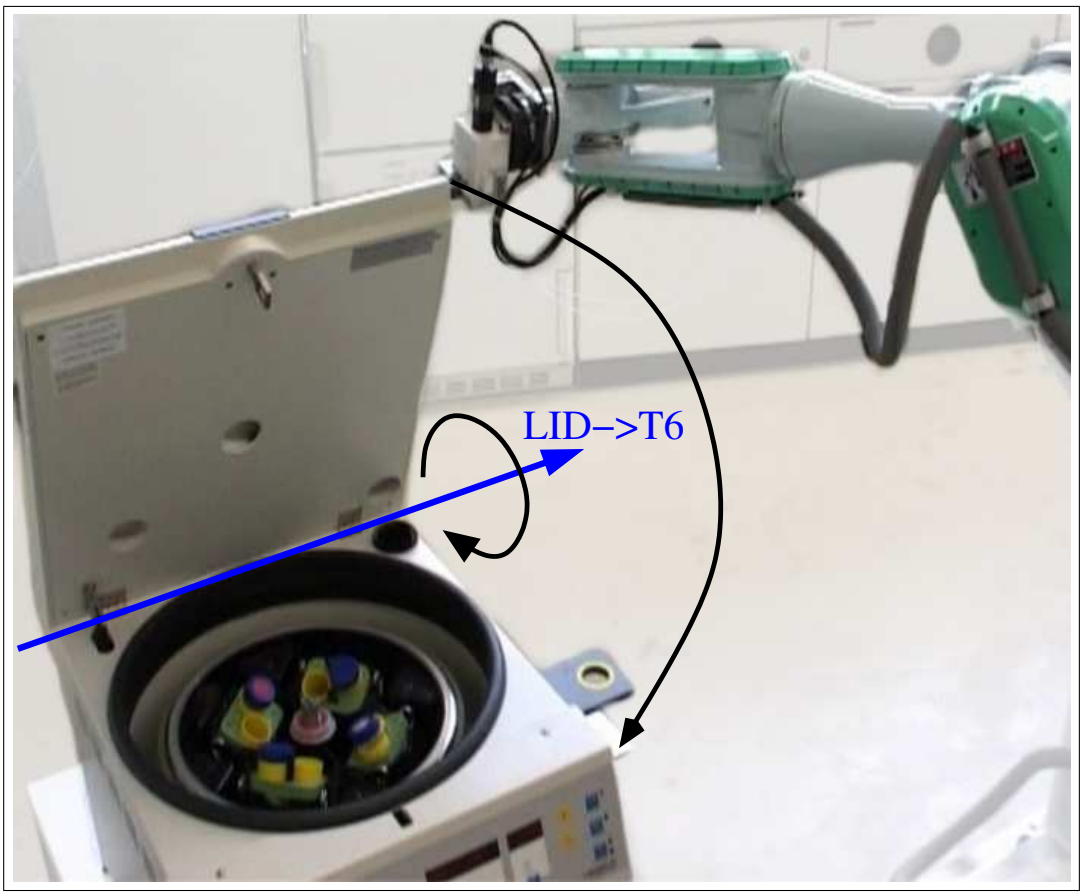


Figure B.14: The hinge of the lid of the centrifuge seen as a virtual robot.

The only thing the user then has to do is to rotate the virtual robot about its z-axis. RCCL will automatically re-evaluate the position equation for the real robot in each cycle and adjust its T_6 accordingly, which – in case of the centrifuge – means that it will move the robot along a circular arc as in figure B.15.

What has originally been implemented in RCCL as the possibility to let two or more robots cooperate can therefore also be used to generate other motions than just straight lines for only one robot. Again, this is impossible to do with the PA-Library because if offers no support for arbitrary motions.

Figure B.15: The robot closes the lid of the centrifuge using a circular motion that is generated automatically by RCCL by passively following the rotation of the lid hinge.

Appendix C

Vision System Documentation

This chapter gives an optional, but more verbose documentation of the vision system used in this work. This documentation consists of a description of the vision hardware in section C.1 and a short introduction into some important aspects of computer vision in section C.2.

C.1 Vision Hardware

In this subsection the hardware used for the colour vision system is described. The hardware consists of two parts: A colour microhead camera and a colour framegrabber board. At the time of the project start this hardware was state-of-the-art.

C.1.1 JAI M1250 Microhead Colour Camera

The "JAI[1] (Denmark/Japan) M1250" microhead (lipstick) colour camera is mounted at the wrist of the arm tool as in figure C.1. The camera delivers raw signals from the CCD chip to a DSP receiver box mounted at the arm base. This box can control the gain and shutter values and can apply a few manipulations to the signal like automatic white balance and backlight compensation. After processing the signal it is then output as either analog colour composite FBAS, SVHS or RGB.

The camera is very compact, as also becomes clear from the specification in table C.1. Its 7.5 mm wide angle lens therefore has only a very cumbersomely adjustable focus and aperture – it has to be unmounted from the tool to be adjusted. The Sensor is very small (8.1 mm diameter) and yet offers almost PAL resolution. This means that the receptive elements on the sensor are very small, which in turns means that the signal has to be amplified a lot in order to obtain bright images[2]. Amplification, however, introduces noise to signals.

Because of this noise level and because of the fact that the signals are transported quite a long way on an analog cable the resulting image quality cannot compete with current digital still cameras. The images in figure C.2 show this rather drastically. With default setup values the colour saturation of the images taken by the JAI M1250 microhead camera are considerably worse than with an Olympus C-300 Zoom digital still camera. Especially low saturated colours like the green boxes are almost completely suppressed on the M1250.

[1] See `http://www.jai.com`, german distributor `http://www.imaging.de`.

[2] Actually, with 752 pixel along 6.5 mm = 8.6 μm/pixel the camera is in the same range as current digital still cameras like the Canon EOS-300D with 3072 pixel along 22.7 mm sensor width = 7.4 μm/pixel, but the post-processing of these cameras is much better today than with the five year old M1250 camera.

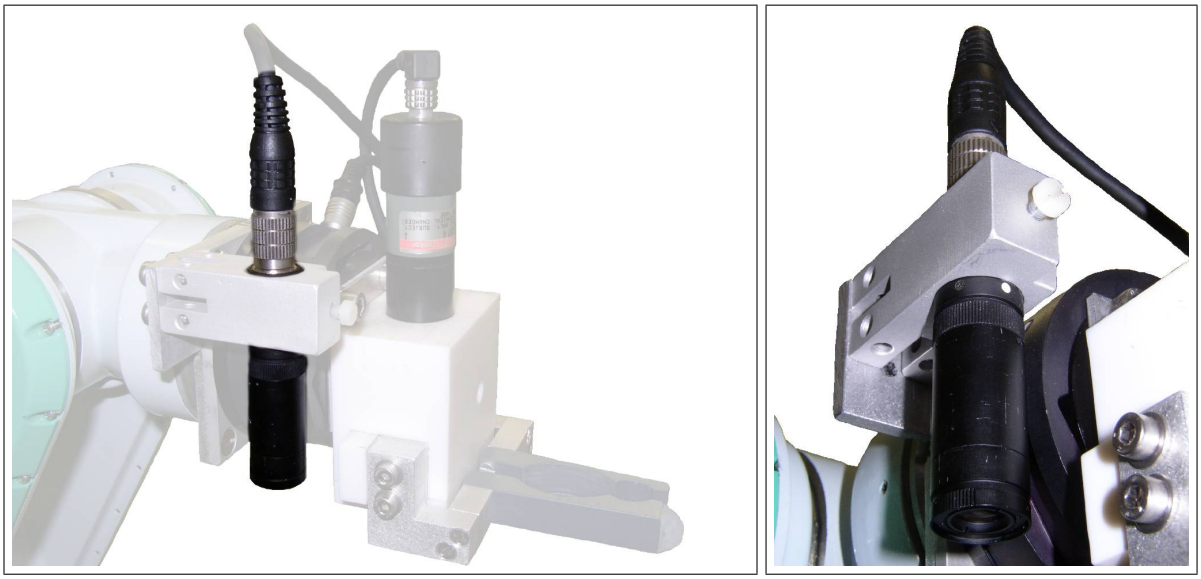


Figure C.1: The JAI M1250 microhead colour camera.

head diameter:	17 mm
image sensor:	6.5 × 4.8 mm (8.1 mm diameter)
	752 × 582 pixel
scanning system:	625 lines, 2:1 interlace
resolution:	450 lines
sensitivity:	1000 lux at f=8.0
minimum illumination:	4 lux at f=1.4
signal/noise ratio:	46 dB

Table C.1: Specification of the JAI M1250 micro head camera.

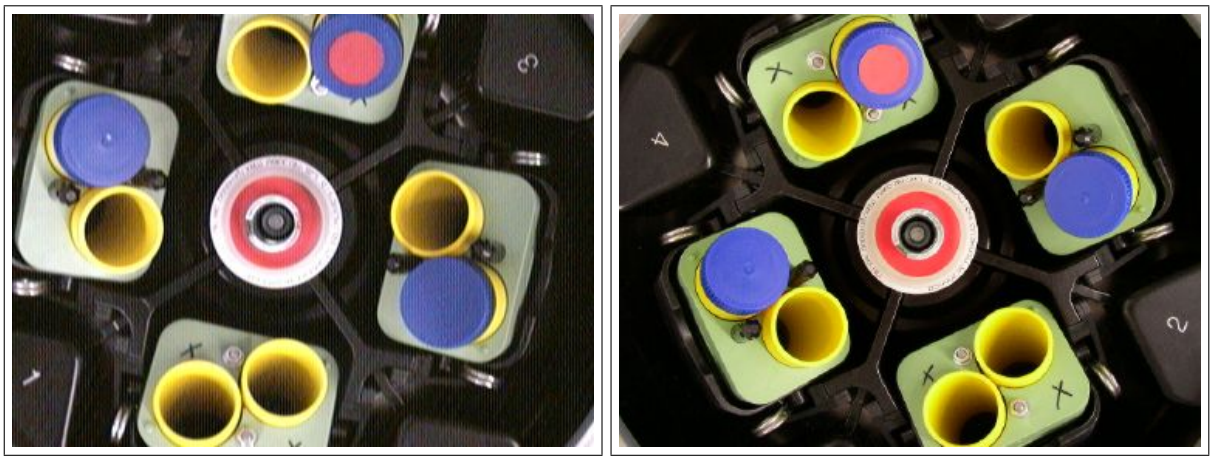


Figure C.2: The colour saturation of the JAI M1250 microhead camera (left) compared to an Olympus C-300 Zoom digital still camera (right).

Another problem with the M1250 camera that can also be seen in this figure is the automatic white balance. The [JAI CV-M1000 Series Operation Manual] says that

> *due to the TTL system employed for Auto White Balance, it detects the highest video level in the image as a white colour.*

This approach is troublesome if there is no true white in the image in the first place, as it does happen in some cases with the objects in the laboratory. In such a case the colour space is shifted so that all colours are slightly wrong. On the other hand not using automatic white balancing at all raises the same problem when the scene is illuminated by direct sunlight through a window, so the vision algorithms have to be tolerant against this.

The usage of a newer microhead camera like the GP-US522(E) by Panasonic[3] can improve the situation significantly by using three separate CCD chips. Also, using newer and better post-processing algorithms in the DSP receiver boxes promise better image quality. Compared to the M1250 camera, the GP-US522(E) has a signal to noise ratio of 60 dB as opposed to 46 dB. The image quality of the GP-US522(E) can therefore be expected to be far better than that of the M1250.

Another interesting idea are digital IEEE1394 "FireWire" cameras, which are available with much higher resolutions because they do not have to use an analog TV video signal as data channel. They process the sensor signal directly in the camera and and can be digitally accessed by firewire, thus also avoiding the error-prone analog transmission. Their only disadvantages are that they are considerably larger than the M1250 camera and that they have not yet been available at start of the project.

C.1.2 Matrox Meteor Framegrabber

The camera is connected via analog SVHS video cable to a Matrox[4] Meteor framegrabber PCI board. It is accessed by means of a linux kernel module. This module does not use the newer *video4linux* interface, but implements a dedicated driver providing access to a `/dev/meteor0` device.

The Meteor can grab colour images in either YUV or RGB mode. Even though the camera's CCD sensor samples RGB values YUV is the better choice because the camera already converts its data to YUV format for transmission over the SVHS cable. Grabbing in YUV mode means that the Meteor has to digitise the SVHS signals, whereas grabbing in RGB mode means that it additionally has to convert them to RGB again. Since every conversion includes a small loss of information due to rounding errors YUV should therefore be preferred.

Using the SVHS cable does provide a better signal quality than using a FBAS cable, but still is subject to noise. This is in particular true on the MP-L655 where not all devices are properly grounded. The image quality resulting from the suboptimal camera, the analog transmission to and digitisation on the framegrabber board and the flawed grounding is rather poor, as can be seen in figure C.3. Vision software operating with these images has to be tolerant against these disturbances. Again, only a digital transmission via IEEE1394 "FireWire" (or USB) could eliminate this problem.

[3] `http://www.panasonic.com/medical_industrial/3CCDColorCameras/index.asp`
[4] `http://www.matrox.com/imaging/products/frame_grabbers.cfm`

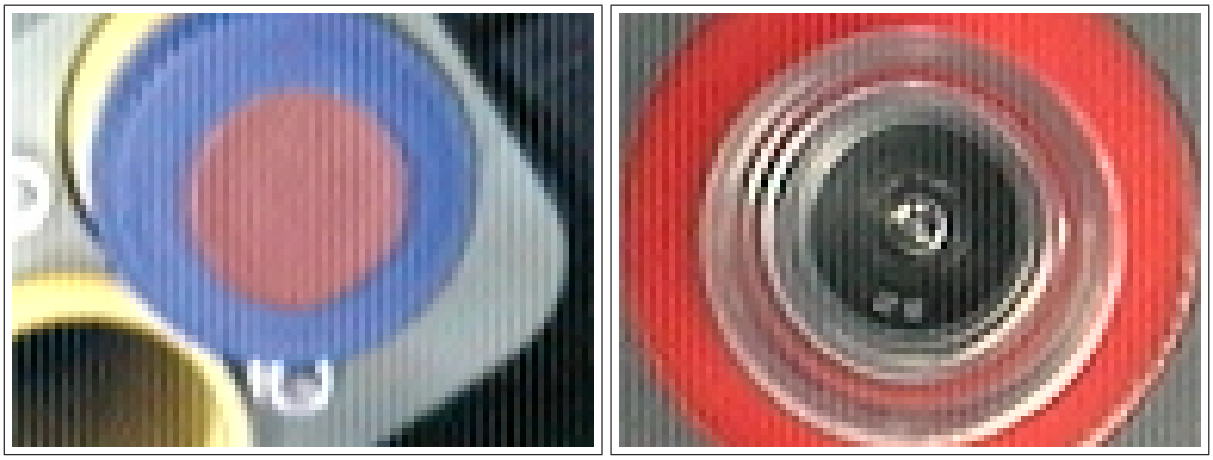


Figure C.3: Sample images demonstrating the poor image quality of the vision system hardware.

C.2 Introduction to Computer Vision

This section gives a short introduction into computer vision and classification. This introduction does not even try to claim to be complete, but only to cover some aspects which are needed or at least helpful for the understanding of the vision system used in this work. As such, it includes some general ideas about image representation (dimensionality), colour spaces, image manipulation, classification and segmentation.

The image manipulation part contains an introduction into basic and well established techniques, even if – like convolutions – they are not used in this work. The reason for presenting them here even if they are unused is is that convolution based image manipulation has a very long history, and the reason for not using them is that they are not sophisticated enough. This of course demands an explanation and hence a separate subsection.

The classification part is split into feature based classification and appearance based classification approaches. The appearance based approaches are not used in the established system, but still deserve some amount of attention because they are an active field of research. They have been briefly tested during this work, but found to be too troublesome. To demonstrate this some results will be given in a separate subsection.

C.2.1 Image Representation

One important question is how to store images inside a computer. This question is obviously not trivial because the representation may have an influence on the performance or complexity of later processing algorithms. The natural way is to look at what is known about how humans work with images like the one in figure C.4.

Humans largely work with images by using spatial relations, that means they use the neighbourship of pixels. They immediately apply this “neighbouring” attribute not only to directly adjacent pixels, but also to those slightly more afar as long as they belong to the same object. The whole question of what an object is is basically centred about what pixels belong together, but humans do not have to explicitly compute these relations because apparently the brain does this automatically at a very low-level stage – they “see” it immediately. This, however, applies only to natural images. If the image in figure C.4 is for example stored as a matrix

Figure C.4: A greyscale image of a euro coin.

$B =$


```
aba9a8a5a8aab1b1abb0a8a5a6acadadada8a6a6a8adaeadb1aeb0b5b3aaacb1b2b3abaeafafacabb5afa6b2a6afa9afb3b1b0abafb5adafaca5b0afb0a7afa7
abaaa9abada8a8a9adadabaeb0ada6a6a7b3b2b3b1b0b3b9b6bdc1c3bcb6bbb2b4bbc6c4c3c3c5bbb1b5b0ada9a1aaa6a5a5a9aaa6a4abaeb4acabaaabacabad
a4a2a4a8a6a8a9a7a4a7a7a8adaeb2b2adb0b5bbb3a59e94796b615d5f5e6361645e5d5c5f687586a4c7cbc9c7b7b3afb3afadabafb0b3adb1aeb6a9b1b4aaa6
a5a6aba7afada9a6a6a6a9acabaaafc4cbc396765953617c9db1c5d5dcdfdfe1e0dcd8d5ccbfaa9070564f6892becfcbbdaaa9adaba3a4a5aeada9ada7b0b3ad
aca4aaaeaea5a5afb6b3b0afb1b8ab8c6e55678cbfd6e0e2ded9dde0dfe3e1e2e0dedcd9d6d9d0d5cdc6b49f7c5a5c8bc8dec9a9a8b6afa4a2afabacafacabae
a7a8a4a1adb5acaaa7a5afb39e7e5c5683c8dbdadadedcdfdde2e2e4e5e6e0e5e2e2e0dfdfdcdbe0ded7d6d1c6c0ae805671b5dccfb0b6b0b3b1a8a8adabacb1
aca8ada9a9a9acadb0b7ad916068a3d2d8d8dcdcdedcdedddce1dbd9d8d6dde0dfdad8dcdadadedddddcdcd9d5d1c4beb8946073b0d3bcb1ada8b0aaaca9a4ac
acada9b0aab4b7b6b9a87f6baed2d4d6d8dadcdddbdcd3d1d7dce2ecf2eeeff0f0efede9ded8d8d4d2d8dbdbd8c9dbc8c1bdbc9b6781cdc7b6b3b3acafb3afa7
a9a7b0b4aca8beb78f76a0d1d4d4d3dcdcd9d5cccfd5e0e8e5e2e5e9edefecdae5e6e7e4e0e1e0dfe3d8d4d1d2c1bbccd2c0c5c5b9946aa5cdb8acaaa7a5aaaf
abaaacaeb1c2b88e88cbd4d8d8d7d8d8d9cbcfd5dbdfe1e1e6e6dfd5d2d7ccbfe6e3e1e3e1dfd8d6dbdedbd4c7babdc5c1cae1c3bdbca96c8accc5a7abaeaaa8
a8acaaa8bfb1939fd0d3d6d9d7d5d1c6cddaddd9bbafaaa4b3c6c8ccced1c9dfe4e3e0dfdedcd7d4d7d7d4d2c7c2bfbdcac9b1c4cfd0c9c0787dc2cfb0aab3ad
a7a8afc2c39faccbd0d4d4d8cfc7bfced8dbdededddcdedfd7d3cedddec9d3e3e3e1e0dedbdcd8d7d8d3d4d0cbb9c0c0b8bdc1bfb1babcc6c18a8bd1b4b4b2aa
b0afb9b8adabc8cdd1d3d6d4cabdccd1d4d7d9dbdddcdee4d3cedce5e2c8dfe0e1e0e0e4dcd6d4d3d3d1d4cdc6bec8c4b9c9c5c3d4cfd8cfc8c88e95dbafacb2
b1bcd2b39dc5c7cdd1ced4cbc2ced2ced3dcd7d9dadadcd8cdd5e9e8c8dbdededfe0dddddbd9d6d8d5d2cfc9c4bdc6c0d3cbc9bdcdc7a5c0c2c5c192a9dbabb2
a7d8bfa2bdc5c8ced5d3ccbec5c9cdcfd0d2d4d9d7dae2ccd1dfe4cad3dcdadadbdbd8d5d4d5dddacfcecbc8c0c1cacfd2c4b1bcbdd1c1bfd1c4bfb1a4cebeae
bacfbbaac0c5c3cfd3ccbcbdc4c7c5cad3d2d1d3d3d9cecbdce6d5c3d5d4d7d6d5d5d3d3d2d2e5d6cdcdccc3b8c1d4dacbbdd3cac4c5c6bca9c7bab7b8b9e1a8
d8c3aeb4bcc2c8cbcbc6b4c5c4cdcbc8c9cccbd4d3d5c5d1e5e5b8d0d2d1cfd5d2d5d8dfcfd1ddcfcbcbcbc2c0cad5d2e4bcc4c3b6b2bebbc7bfbcbab8c3d4bb
d3d6b6aeb3c1c5c7c6b2b5bcbfc1c5c3cac9c7c9cfbac6e1e6c0c3cdc9c9cbcad0dbe2d1cecdecd2cccfcecab8c4d0d7bbc4b3c1b4bdb6c3c3bdbbb9b1bfcac9
cbdeb7b6bec7c7c4bfaeb9bebdbdc0c5c3c5c2d6c0bfcddfc4bac2cec4c2c4c8c8a9abbcd5e0ddd4cccfc9dac8c5ccbdc0c3b2c3b4bfb6ccbfb3b9b7bebfc6d0
bde8bdb8c0c5c5beb6aebbbdc0bfbfbfc4c0c5c4c2c9dbd1afbac8c4bec1bfc4c5c4c4bcb4c1c3c9ccd0d3d1d5d1beb3c3b1c1b4b1b2bcd0bdb6b9b9babcd1cf
c1edb9bac1c5bfbebaafc0c5c1bfbbbcbebecdbfcad7ddb7aeb1b4bcbcbbbdc0bfbec1c4c5d0d2c1c2d5d1d9d1d7bab4afb1b1b5ada7c2c4bcbab4b7bdbdd1d5
bdefb8b9b8bac1beb7abbebfbcbabbb8b6c1b5bfc6c9b6b4afafb8b6b3babdc0bdbcb7b3ccc3c8e1cbc8cebdb5babbb8b9d1c2c2c2b0b8b7b8beb8b5bebecdcb
c8f4bbb8b3b8bebcbbacb4b7bab9b7b4bcc2b4bcc6b7adaab1a8a8acb3c5cecdb8b8b9c4cebdb3bcd6d5cbc5cec5c4acbdcb9ec3c7a8bfc3a6afafd5c0bbcfa7
c5ecceb6b7b7b8bbb9abadb4b3bab2b4c4babbbbb7acabaaaca7abadb4d1adbed2c8d0beb5cdc9ced1c6bfcdd4d4b7b4caadc7afa2b6ccb5beb0bed8c2b5d288
a0dde2bcb1b5b5b9b2ada2adb5c1d0c3b3b2b4b6a7c6a6aaa9aca6abd5b8beb9bec3c3ceb9afb7b9cad8b6bcb198abbdcdaca6d9b1b2c9c3b6bebebbccbfd77c
80b7f0beafb1b4b5b9a5bbb6c1bfbbb6b7bdbec4aac0c8a8a4a7a2b2c3c3c6bec1c4a6989ca39aa2a4b9d9c39faeb4a291b5b191a6b39dcfc4c2a6b0bfd89174
5594e1ddb2b2afb2b0a4a5828f8c868e8f91968f869796adb3aeb59a9eb6bdc9b398a3a5adb9b3b7c8aaacd0d4aebab3b9bcc2b3b8bcc4b0aeb0bbc0bea45f75
4b65a4f2c4b1adaeaaa6aca896adafaeaeafaea4a7a99f9b928a959b979f968d94acb3b3afafadaba3b4a398bbcdbcb5bec69cc0bbbbbecdceb8b9bbad6e5c70
434e69c3e1c1aeaaa8aeabb2a598aeb1adb3b5aca299a3a2a3a5a1999daaa3a5a7a8a7afafaaaeaeb3b6cecaaba2c3b6b8b4abc8bcbdb7b8b7c9bcbc8d635e73
46455478d9d4beb2b3adafa6a1ad96a4adafb9b4ac9ea29da3a6a4afa6b5b2acbeb9aba8abaeb2aaaeb29cb0adb1c0bdb2acc3c5bec7b6b7bac2bb9f62575975
513f4a537ad8d1b4aba8b4aaabaeac9e92a1a8aba89eaaa7aaa3a2a6a7a499a19abdb7b2b0b2b3b7b7bab1bbc9b7abacc6c0c3afb5b8b4b5bbbe934c4e4f587f
5b3e4547597bd3cbbbb1b5aeababa9aca29a95a7acaea6a3aaa8a2aba8a6aeabafbbb7b6b3b3b7bcc1c1c3a6b7bcbfbabbada7b4c2bfb5b9b4783c45494e607e
654e414043586fadd2c3b2aba8a6a9a8aca4aa9c9b98a0a7a9a6a6a5adadafafb4bbb9b5b5b8c1c5acb4b7b6b9babdbcc5b7c5b4b5c0d2a34e383a45464c6f89
635740393a3c525a72ced1baacacada4a7a0a59a9ca2a3a09e999ca29d9da4a8aca8aab0acb4b5b1bac0b8b5bbceb3b5b5bbbcbfc9cc6b2e393b383e3f557089
675c4d3c3538393f545882cfc3b2afb0a89f9f9c989e9ea9a7b0afb2adafb1b3b1b6b4b1b9babcc7c9b9ccc5b1aeb7b6b8babfc5974e313c3736343648627c92
6f5e5a473634353538 48595b8bc6c3beb4aea8a199a19ca8a6a6aab1afadb3b8b8b9b9b7b8b7b2a9adafafbcbcb9b6bec5c98446303c38333233303e55687f8d
77615751433432333132364b544973b8d5bcafa9aba8a4a8a4aab1aea8adb5b4b7b9b9b8b8b4b5c2b7abb1bfb7cecfac623238483a2f2d2f2c2a384d5c6a7685
7c685f554d41332f2f2d2d3437434c464f7fafc5c2b4b1b0b0b7b2a2a8a9b2b0b2b5b3b7b9bbb1afb6becccaa26b462c3a3d34322f2c2a282934445764768394
8573665d534740352e2c2b2d2e2f2f363e423738506e8a9baebec2bdbcb8bec4c2bec2bec1b7a59b806342302d3537352f2b29292625262c3741505f72789494
86726b62554e47433b312c2e2b2a2b2b2c2f3134343532312c30313a4b4a5254534e463d322b2b33353936302f2c2928282623232127313943 4c56687c868899
9089776f5e574b49464037312a262627262828292b2b2b2d3333323331322f303132343332312f2f2a2b292726242422221f1c222e3338424c556777828ca097
8b83746f6c6057514a44423a342c2625262525272525272526282928282627262726262726262525242423222221211e1e20232b33333c454e5861707c85979295
8d868484756c5e57524d46403c38352e2a232324222222222222242323211f2121212221232222221f201f1e1e1c1d20252c3234363c46505a646c79889586979e
9398918b86786b6359524f47433e3c3935322c2626221f1f20201f211f1d1e1d1d1d1e1f1e1d1d1c1c1a1b1f25292d3335383c4249505e68727b8490989d998f
93948d8a827d7a6e6a5c56534b4542403a39353333312f2b29242222222220201f2020202022242425262a2b2f3337383a3e43474e565f697d878f9692959da0a3
988a95908e847e727565645d574d4849433f3c393635343432302d2b2f302e2c2d2c2e2d2d2e2f2f2f34383a3c3e4144474f585f66777c8587878f9fa1979f9d
999f948f9a90858c83766e6a635a55514e4946413d3d3a37363534333233323232323333233336383a393e414143474c50575d6a6e72838a8f94989a96a0a19aa7
a69ea49a959d8e8a8789837c6e69635e5f554f4b494742403f3f3b393738393a3a3a3c3d3f40414445484b4d5153585e656c7d7e84928e91a0a09f9a9ba2a9a2
```


of hexadecimal 8-bit greyscale values in order to preserve as much of the spatial relationship as possible a human finds it much harder to make anything out, although both representations contain exactly the same information.

On the other hand such a matrix does not help a computer a lot. It assumes that a computer can draw the same benefits from such a representation as a human, which is not true. A computer always has to explicitly compute relationships, and whether it does this by means of going along rows and columns of a matrix or any other data structure makes no fundamental difference. Even the fact that a matrix is a 2-dimensional structure does not matter, and so in this work as well as in many others images are for simplicity treated as 1-dimensional structure, as a vector

$$\vec{b}^T = \left(\texttt{aba9a8a5a8aab1b1abb0a8a5a6acadadada8a6a6a8adaeadb1aeb0b5b3aaacb1b2b3abaeafafacabb5afa6b2a6afa9afb3b1b0abafb5adafaca5b0afb0}\ldots \right)$$

instead of a matrix B. This is perfectly legal as long as the width of the original image is stored elsewhere too, because the two representation can be transformed into each other with the equation

$$B_{[x,y]} = b_{[y \cdot \mathrm{width} + x]}$$

The dimensionality of the space spun by such a vector is independent from the fact that it can be written as a 1-dimensional structure. In fact, image vectors are amongst the highest dimensional vectors used in computer science. The table of numbers of pixel (number of elements of the vector) for the usable resolution of the most prominent video formats can be seen in table C.2.

norm	resolution	pixels (gray)	pixels (colour)
PAL (half)	384×288	110592	331776
NTSC	640×480	307200	921600
PAL (full)	768×576	442368	1327104
HDTV (1)	1280×720	921600	2764800
HDTV (2)	1920×1080	2073600	6220800

Table C.2: Usable resolution and number of pixels of the most prominent video formats.

In case of the full PAL resolution of 768×576 pixels this means that an image carries 442368 pixel in greyscale format or 1327104 pixel in colour format. Additionally taking into account that the pixel values are usually sampled with 8 bit precision in a computer this means that in a full PAL resolution colour image one of $256^{1327104}$ or $2^{10616832}$ possible situations is encoded. Considering that the currently biggest supercomputer[5] has "only" a little more than 2^{43} bytes of memory simply storing for later comparison in order to detect objects is utterly impossible (and this even completely ignores the question whether it is feasible to obtain the necessary set of teach images in the first place). Clearly, vision algorithms which drastically reduce this dimensionality are required.

C.2.2 Colour Spaces

Colour information is – contrary to greyscale information – not unique. From the different colour models or colour spaces that do exist the ones used in or interesting for this work shall be discussed in this subsection, taking the colour image in figure C.5 as an example. Most of the information has been gathered from the Colour FAQ by Charles Poynton[6].

According to table C.2 colour images have three times the number of pixels than greyscale images. An alternative point of view is to say that they do actually have the same number of visible pixels, but that for each pixel the colour information is encoded in three separate channels[7]. The motivation for these channels directly leads to the first colour space.

C.2.2.1 RGB

The RGB (*red*, *green* and *blue*) colour space is used for example in computer and television displays (but not in (analog) television signal radio/cable transmission). It appears to the "natu-

[5] The "Earth Simulator" in Yokohama, Japan, has 5120 CPUs with 2 gigabyte memory each, totalling to 10 terabyte. See `http://www.es.jamstec.go.jp/esc/eng/index.html` or `http://www.top500.org`

[6] See [Poynton 2002].

[7] Other colour spaces like the 4-dimensional CMYK used for printing exist, but are not of interest for this work.

Figure C.5: A sample colour image.

ral" colour system because it is motivated by the human eye. The human eye uses two different major types of light receptors: rods and cones. The about 120 million rods are very sensitive to light of all wavelength and are used for greyscale vision. The about 6 to 7 million cones are far less sensitive to light, but have a more selective sensitivity for light of certain wavelengths and are used for colour vision. They again can be separated in three types[8]:

1. "Blue" cones have a sensitivity in the area ranging from 400 and 540 nm with a peak near 419 nm.

2. "Green" cones have a sensitivity in the area ranging from 450 to 640 nm with a peak near 531 nm.

3. "Red" cones have a sensitivity in the area ranging from 550 to 700 nm with a peak near 558 nm (which is actually more like yellow).

Figure C.6 shows a (very) rough sketch of this spectral sensitivity of human eye cones.

The RGB colour space therefore defines three colours *red*, *green* and *blue* and uses them in additive colour mixing e.g. for *cathode ray tube* (CRT) displays, or their inverses *cyan*, *magenta* and *yellow* (CMY) in subtractive colour mixing e.g. for printing. Both colour mixings can be seen in figure C.7.

[8]Which makes up for a total of four different receptor types in the human eye even though the colour information is encoded by only three of them. This situation is similar to the CMYK colour space used for printing where it is theoretically sufficient to use ***c****yan*, ***m****agenta* and ***y****ellow* (CMY) to print all possible colours, but *blac****k*** is additionally used to generate a deeply saturated black.

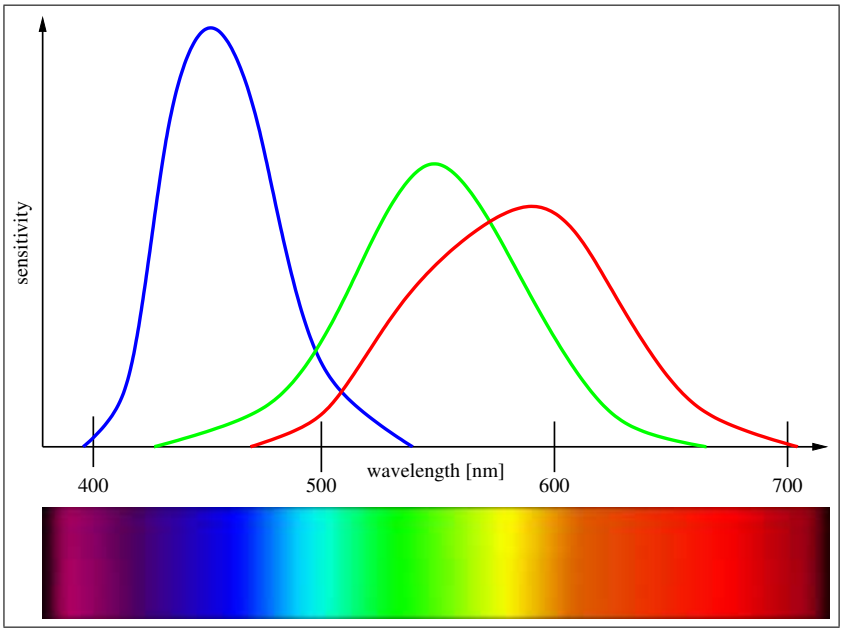


Figure C.6: Approximate spectral sensitivity of the cones in the human eye. The sensitivity of the "red" cone lies more in the yellow.

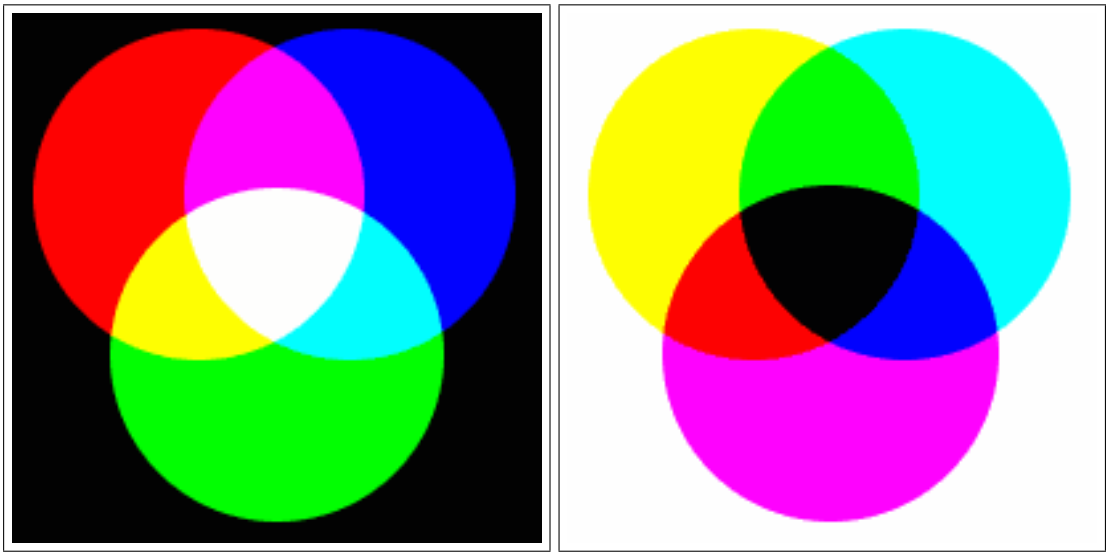


Figure C.7: Additive mixing of RGB colours (left) and subtractive mixing of CMY colours (right).

Using the RGB colour space the sample image in figure C.5 can be decomposed into the three channels *red*, *green* and *blue* as shown in figure C.8. Note that in each of these channels the intensity of one basic colour is encoded separately. This means that if the overall intensity (brightness) of the scene changes the values in all three channels will change too. A vision system that is supposed to be invariant against changes of the illumination may find it difficult to deal with this, but then other colour spaces exist that do not share this feature.

Figure C.8: Decomposition of the image in figure C.5 into (from left to right) the R (*red*), G (*green*) and B (*blue*) channels of the RGB colour space. The channels themselves are by definition greyscale images, but have been re-colourised here for better visualisation.

C.2.2.2 YUV

One alternative is the CCIR-601 YUV colour space[9] used in analog video radio/cable transmission. It was introduced with the invention of colour television as a result of the desire to keep it compatible to black/white television. By defining luminance

$$Y'_{601} \quad = \quad 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B'$$

and encoding the colour information (UV) separately it allowed old black/white television sets to display the luminance only while at the same time the new colour television sets could use the full information[10]. For displaying YUV images on a RGB monitor the two colour spaces can be converted into each other by

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} \quad = \quad \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} + \begin{bmatrix} 0.25678823 & 0.50412941 & 0.09790588 \\ -0.14822290 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778832 & -0.07142737 \end{bmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

and

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad = \quad \begin{bmatrix} 1.164383574 & 0.000000000 & 1.596026777 \\ 1.164383574 & -0.391762275 & -0.812967628 \\ 1.164383574 & 2.017232129 & 0.000000000 \end{bmatrix} \cdot \begin{pmatrix} Y - 16 \\ U - 128 \\ V - 128 \end{pmatrix}$$

using computer-utilisable values in the range of 0..255 for each component. As follows from these equations the Y component has an excursion of 219 at an offset of 16 (usable range: 16 (black) to 235 (white)) and the UV components have an excursion of ± 112 around an offset of 128 (usable range: 16 to 240). Values outside these ranges will lead to values outside the 0..255 range for RGB and may occur by using inaccurate conversion routines, but are never delivered by compliant video devices.

[9] CCIR – Consultative Committee for International Radio; a predecessor organisation of the ITU-T/R.

[10] The prime symbols denote the usage of non-linear, gamma-corrected values. The subscript 601 denotes the CCIR 601 recommendation, describing CRT phosphors used at the time of introduction of NTSC colour television in 1953. Contemporary CRTs use different phosphors and hence the newer CCIR 709 recommendation defines $Y_{709} = 0.2125 \cdot R + 0.7154 \cdot G + 0.0721 \cdot B$.

Using the YUV colour space the sample image in figure C.5 can be decomposed into the three channels as shown in figure C.9. As can be seen that the UV channels are very pale, low-contrast images. Only a fraction of the usable range of 16 to 240 seems to be used by most pixels, resulting in a mainly neutral gray image. This is because "being coloured" still allows for a variety of colour saturations. Colour saturation is expressed by the euclidian distance in the UV plane from the neutral (128, 128), and the fact that most pixels are very close to this value shows that most colours in the image are of very low saturation. Looked at from an abstract statistical point of view the UV channels do by far not carry as much information as the Y channel. This is consistent with the human colour perception which is very sensitive to changes in the colour – it does not need much information to tell colours apart.

Figure C.9: Decomposition of the image in figure C.5 into (from left to right) the Y (*luminance*), U and V (both *chrominance*) channels of the YUV colour space.

This aspect of colour saturation can also be seen in another visualisation that is used later in this work. Figure C.10 shows the UV plane with the colours used by the image in figure C.5 marked in black. It can be seen that the pixel are aggregated around the image centre with only a few branches going into the outer regions towards saturated colours. On the other hand it can also be seen that these branches go into the blue, green, yellow/orange and red colours that the image basically consists of.

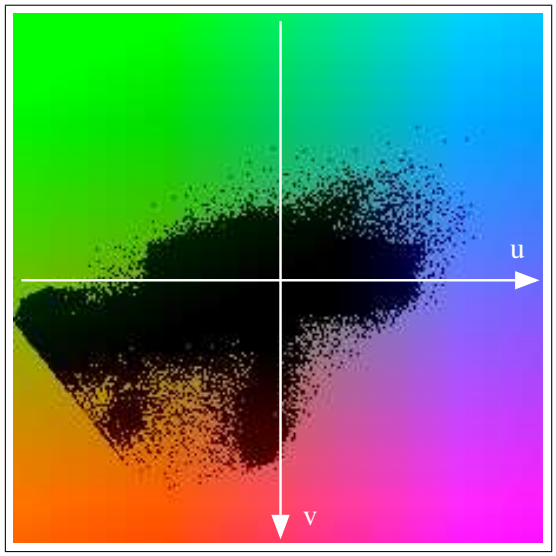


Figure C.10: The UV plane with the colours used by the image in figure C.5 marked in black.

For this work YUV has been chosen as the low-level colour space because the vision hardware using analog video transmission natively delivers YUV image data rather than RGB, so that no additional conversion is needed. Higher levels operate with notions which are not completely identical, but very similar to yet another representation: The HSV colour space.

C.2.2.3 HSV / HSI / HSL

In the YUV colour space the image *luminance* is separated from the *chrominance*, but it has been shown that this chrominance still mingles the colour hue and saturation. As a natural consequence other colour spaces have been defined. One of these colour spaces is the HSV colour space invented in 1978 by Alvy Ray Smith[11].

The three channels of the HSV colour space represent the *hue*, *saturation* and *value* (luminance) of a pixel. The conversion goes along the following lines[12]: In the UV plane a colour is given by a vector (u, v). Using polar coordinates this vector could also be described as $(d = \sqrt{u^2 + v^2}, \alpha = \arctan \frac{v}{u})$. The HSV colour space does basically exactly that, naming the angle α the *hue* and the distance d the *saturation* of the colour.

This transformation of the YUV colour space leads to a cylindric representation of the HSV colour space. In this cylinder it is theoretically possible to use several distinct hue and saturation values in order to obtain the same colour, for example the H and S values do not matter in case of V=0 because black is black and remains black. As a consequence of this the HSV colour space is more usually represented by a cone, where distinct numerical values really represent distinct colours. Both of these representations can be seen in figure C.11.

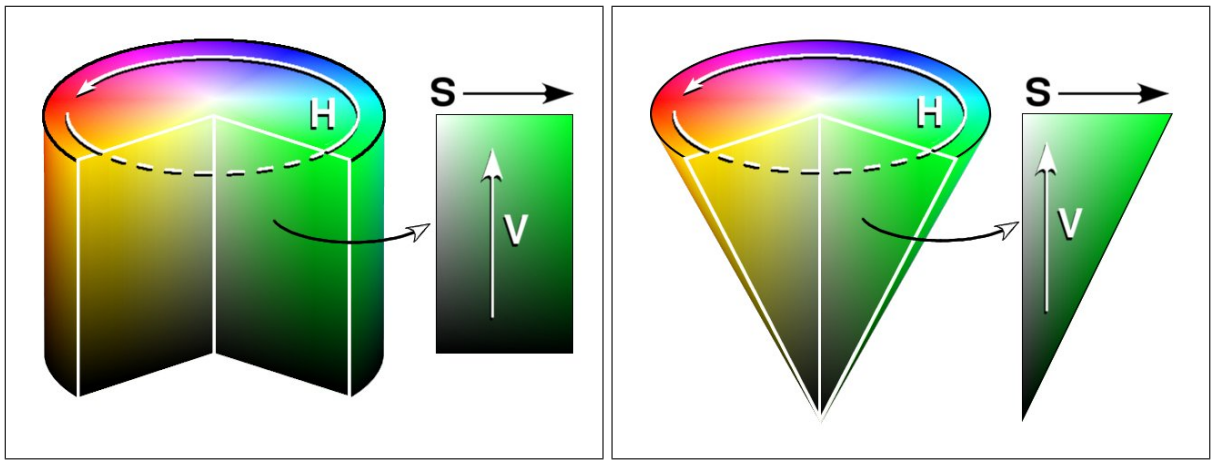


Figure C.11: The HSV colour space, represented as a cylinder (left) or cone (right).

Since the HSV colour space closely resembles the human colour perception it is often used in computer graphics. In these applications a "colour wheel" as in figure C.12 is used to determine colours. With this wheel a user can intuitively enter the 3-dimensional colour information by using an only 2-dimensional input structure.

[11] See `http://alvyray.com`.

[12] The *exact* conversion equations are a bit more complicated. They are not given here because they are not needed in this work. This work does use the notions of colour hue and saturation, but does not require them to be compatible to the HSV system because these values are only used internally.

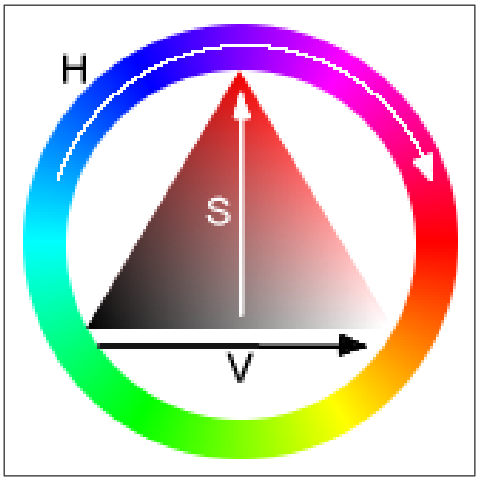


Figure C.12: The HSV colour wheel.

Using the HSV colour space the sample image in figure C.5 can be decomposed into the three channels as shown in figure C.13. All three channels are by definition greyscale images, but the H channel has been recoloured for better visualisation.

Figure C.13: Decomposition of the image in figure C.5 into (from left to right) the H (*hue*), S (*saturation*) and V (*value*) channels of the HSV colour space. All the channels are by definition greyscale images, but the H channel has been re-colourised for better visualisation.

Variants of the HSV colour space include the HSI (*hue*, *saturation* and *intensity*) and HSL (*hue*, *saturation* and *lightness*) colour spaces. Contrary to the HSV cone (or hexcone), where white is in the middle of the plane of colours with full luminance at the base of the cone they both use double cones (or double hexcones) put together base-to-base. The two apexes of the double cone correspond to black and white. and the most saturated colours are in the middle the of the I/L axis where the structure is widest. As such both HSI and HSL more closely resemble a deformed version of the RGB colour cube with I/L being the diagonal between black and white. Their only difference is in the interpretation of the I and L value.

C.2.3 Linear Image Filtering

An important processing step in computer vision are image manipulations known as filters. These filters can be used to improve image quality by applying certain smoothness criteria to remove distortions or to detect low-level image features like edges. They can be divided into

linear and non-linear filters. In this subsection the basics and some examples of linear filters shall be given, and in the next subsection some examples of non-linear filters.

C.2.3.1 Convolutions

Linear filters define a *mask* (also called *core* or *kernel*)

$$H_{k,l} \in \mathbf{R}^{(2m+1)(2n+1)},\ k = 0, \ldots, 2m,\ l = 0, \ldots, 2n$$

that describes a local linear operation in a certain environment around a pixel of the original image. The size of the mask can vary, but is usually small and square (3×3 up to 7×7) because for these sizes special hardware exists which can apply them to video signals in real-time. In general, the new image is computed by applying the mask h to each original pixel $f_{i,j}$ successively as per

$$g_{i,j} \quad = \quad \sum_{k=-m}^{m} \sum_{l=-n}^{n} f_{i+k,j+l} \cdot h_{m+k,n+l} \tag{C.1}$$

Care has to be taken at the borders of the image because parts of the mask may reach outside the image. In these cases several strategies are possible:

- Apply the mask only to those pixels where it fully fits into the image, explicitly meaning that it is not applied to a certain margin of the image. Depending on the operation either the original pixels can then be copied into the margin of the new image or the border can be dropped from the new image, making it smaller.
- Apply the mask to all pixels leaving no border and assume those parts where the mask exceeds the image to have a certain fixed value (zero). This strategy maintains the original image size, but may lead to unusable results in the new image margin. For example, in case of smoothing where the sum of all pixels is divided by the mask size it will make the image the darker the closer it bets to the border because more pixels are assumed to have a value of zero.
- Apply the mask to all pixels and wrap at the borders, that is, use pixel values from the opposite borders. This strategy is only recommendable for recurring patterns, which is rarely the case in real-world examples.

Actually, equation C.1 is only a specialised version of the general equation

$$g(t) \quad = \quad \int_{-\infty}^{\infty} f(t - t')h(t')dt'$$

known as the *convolution* $f(t) * h(t)$. In the time domain this is a bit awkward to write, but using the Laplace transformation

$$\mathcal{L}\left(x(t)\right) \quad = \quad X(\omega)$$

it can be rewritten as

$$\begin{aligned} \mathcal{L}(g(t)) &= \mathcal{L}\left(f(t) * h(t)\right) \\ &= \mathcal{L}\left(f(t)\right) \cdot \left(h(t)\right) \\ &= F(\omega) \cdot H(\omega) \end{aligned}$$

The convolution operator "$*$" in the time domain becomes a simple multiplication "$\cdot$" in the frequency domain.

C.2.3.2 Smoothing

One example of a convolution filter is the *mean* filter used for smoothing an image. Its filter kernel is given as

$$H \in \mathbf{R}^{m,n} = \frac{1}{mn} \begin{bmatrix} 1_{1,1} & \cdots & 1_{1,n} \\ \vdots & \ddots & \vdots \\ 1_{m,1} & \cdots & 1_{m,n} \end{bmatrix}$$

or, in case of $m = n = 3$ as

$$H = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The filter obviously adds up the values of all pixels covered by the kernel and then returns the average of these values. As such, it smoothes outliers from the image, but does not remove them completely. Also, it does not differ between noise and meaningful image contents, therefore it blurs the image.

Examples for the mean filter can be seen in figure C.14. The source image has deliberately been chosen to be noisy. As can be seen, by averaging over the pixels in the filter kernel the filter effectively suppresses noise, but also blurs out sharp edges. In order to smooth images while not overly blurring out the image other filters are therefore needed.

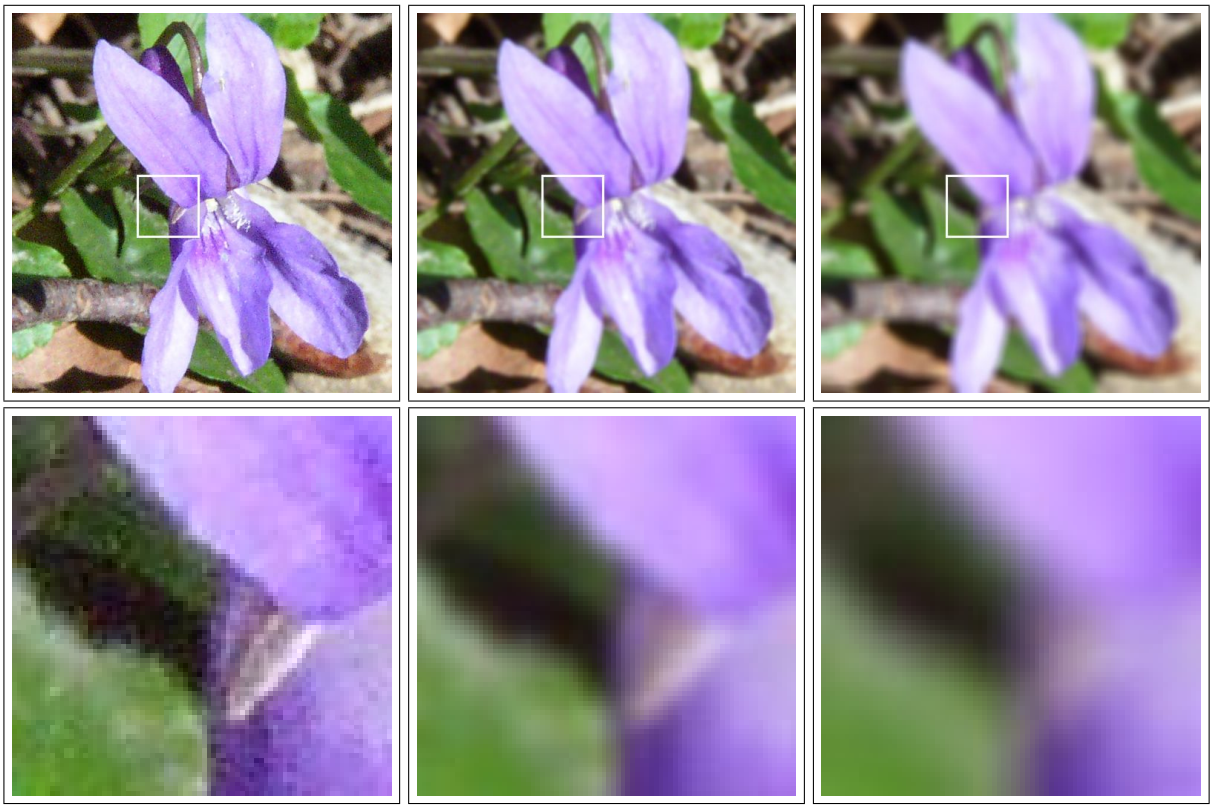


Figure C.14: The mean filter smoothing an image. Upper row from left to right: Original image, 7×7 mean and 15×15 mean. Lower row: close-ups of the areas marked in the upper images.

C.2.3.3 Laplace Filter

The Laplacian $L(x, y)$ is a function defined to be the sum of the second order partial derivatives in the x and y directions of some other function $f(x, y)$.

$$L(x,y) \quad = \quad \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Because of the use of derivatives it is sensitive to changes in the original function, which in image processing terminology means it can detect image edges. It is not sensitive for the spatial direction of an edge, but it is sensitive for the gradient of the edge.

Unfortunately, derivatives in general and the second derivative in particular are very sensitive to noise. Already a little bit of noise in an otherwise uniformly filled image area can therefore be treated as a corner by the Laplace filter. As a result the Laplace filter is difficult to use without prior image smoothing.

For example, a typical 3×3 Laplace filter kernel of

$$H_{\text{Laplace}} \quad = \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

will produce results as in figure C.15. The Laplace image has an average gray level of 128 with edges being either darker or brighter, depending on their gradient. The output images have been contrast-maximised, yet they have a very low visible contrast. This is because single spikes caused by noise use up the full output scale so that real edges are not so clearly visible. Only if the source image has been smoothed in advance the edges can be seen, but immediately start to get blurred due to the smoothing.

Figure C.15: The Laplace edge detector applied to a greyscale image. From left to right: Original image, and Laplace output of original and smoothed image. The Laplace images are contrast-maximised.

C.2.3.4 Sobel Filter

An alternative way to detect edges is the Sobel filter[13], which only looks at the gradient of the image, but independently for several spatial directions. Several approaches are possible, but the most common implementation uses two separate kernels for x- and y-direction.

[13] See [Sobel 1990].

These kernels are not as sensitive to noise as the Laplace kernel because of two reasons: First, they use only the first derivative along a single direction, for example a 1-dimensional kernel

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

in case of the x-direction. Second, it uses an implicit smoothing perpendicular to the derived axis, again, for example

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

in case of the x-direction. Combined, this makes for a filter kernel of

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

for x-direction direction and

$$H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

for the y-direction. With these kernels both the norm

$$g = \sqrt{\frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2}$$

and the spatial direction of the edge

$$\alpha = \arctan \frac{\left(\frac{\partial f}{\partial y}\right)}{\left(\frac{\partial f}{\partial x}\right)}$$

can be computed. Figure C.16 shows an example of the Sobel filter. Only the strength of the edges is shown, not their direction. Contrary to the Laplace filter the edges can be clearly seen without prior smoothing of the original image, indicating that the Sobel filter is not as sensitive to noise. However, also contrary to the Laplace filter, in this implementation the direction of the gradient in brightness space can not be seen due to usage of the norm.

C.2.4 Non-Linear Image Filtering

The results of the above linear filters are not in all aspects fully satisfying, however, alternatives exist with the non-linear filters. Non-linear filters are all those filters that cannot be expressed by a simple convolution because they include – maybe in addition to a convolution – non-linear steps.

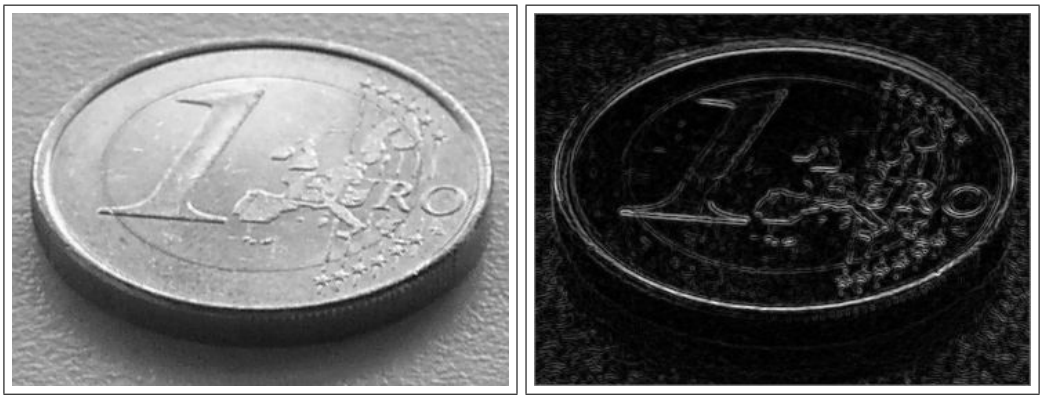


Figure C.16: The Sobel filter applied to a greyscale image. The left image is the original image and the right image the edge strength as computed by the Laplace filter. Contrary to the Laplace filter the edges can be clearly seen without prior smoothing of the image.

C.2.4.1 Median

The median filter is a smoothing filter which, instead of computing the average of a set of pixels, computes the median. This median is the average of the set of pixels with some extreme outliers removed in advance. Technically, this often done by maintaining a sorted list of pixel values, discarding some elements at the beginning and end of that list and computing the average of the remainder. Since sorting is a non-linear operation the median filter cannot be expressed as a convolution. Due to the explicit removal of outliers the median filter produces a smoother output than the mean filter, but since it still blurs the image no example is given here.

C.2.4.2 Smart Smoothing

Another class of smoothing filters use what is called here "smart smoothing" techniques. The idea with these techniques is that smoothing should only be applied to those areas of the image where it can be expected to be helpful. For example, areas uniformly filled with a colour but slightly distorted by noise should be smoothed, but edges between areas of different colours should not. As with smoothing in general there is not *the* smart smoothing filter, and the smart mean (*smean*) filter developed for this work is just one possibility.

This smean filter is similar to the mean filter in that it uses the same kernel to compute the average value p_μ of an environment of pixels, but differs in the step that comes afterwards. Where the mean filter simply assigns this average value p_μ to the new pixel p_{new} the smean filter also computes the deviation σ of the environment of pixel. This σ expresses how "equal" the pixels in the filter kernel are and is low when the pixels are similar and high when they are not, as in case of an edge. Together with a scaling factor c it is then used to weight the average and the original pixel value to compute the new pixel value as

$$p_{\text{new}} \quad = \quad e^{\frac{-\sigma}{c}} \cdot p_\mu + (1 - e^{\frac{-\sigma}{c}}) \cdot p_{\text{orig}}$$

Figure C.17 shows an example of the smean filter. Compared to the mean filter in figure C.14 it can be seen that the smean filter only blurs those areas of the image where pixels have already been somewhat similar, but maintains rather sharp edges. This feature makes the smean filter very interesting for smoothing a YUV colour image where the UV colour plane has a rather low signal/noise ratio but edges should be maintained by all means.

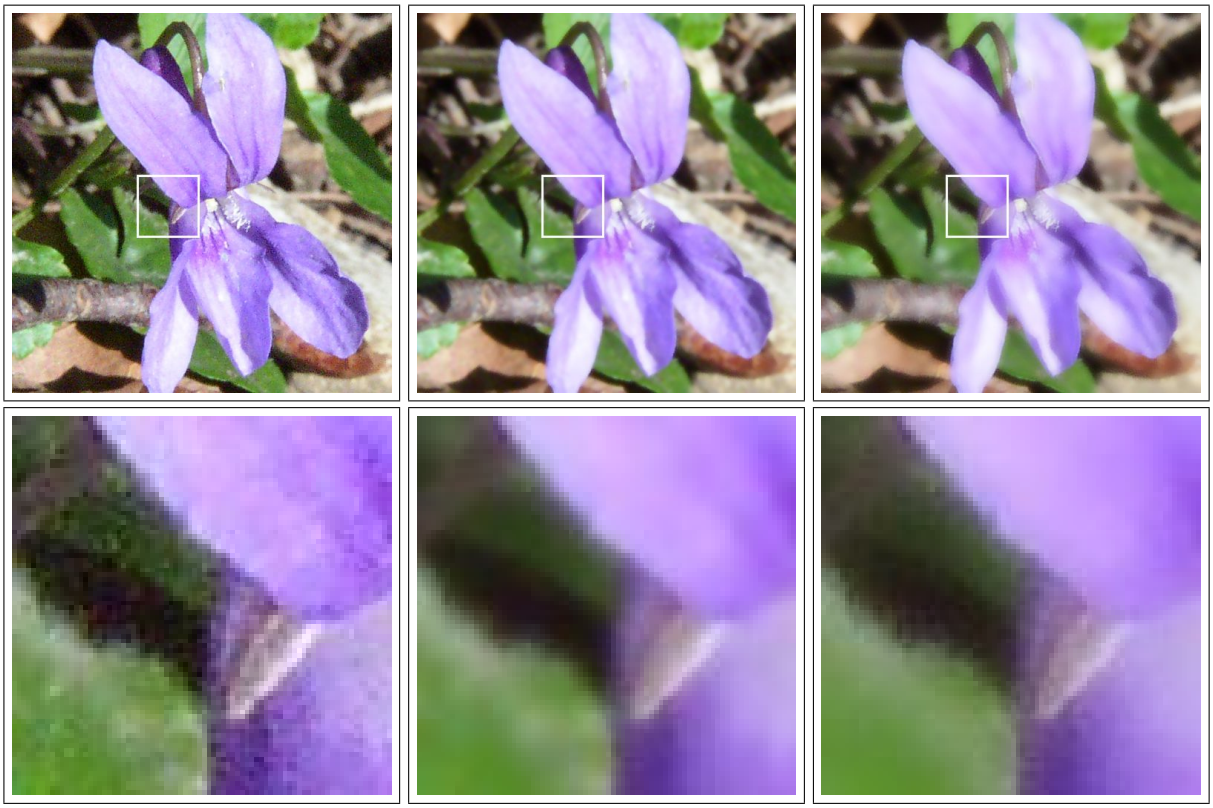


Figure C.17: Smart smoothing of an image. Upper row from left to right: Original image, 7×7 smean and 15×15 smean. Lower row: close-ups of the areas marked in the upper images.

C.2.4.3 SUSAN

An example of a non-linear edge detector is the SUSAN (*smallest univalue segment assimilating nucleus*) filter by Smith and Brady[14]. The SUSAN filter places a circular kernel over a centre pixel and then computes the size of a region of pixels inside the kernel with a value that is "similar" to the value of the centre pixel (the USAN). An illustration of this principle is given in figure C.18. As can be seen the size of the USAN corresponds to the distance to an edge: Both in- and outside the object the size is at its maximum, while at the edge it drops to the half (in case of corners it would drop even to a fourth). The size of the USAN can therefore be used to detect edges without derivating the image data. This is an important feature because by explicitly avoiding derivations the SUSAN filter is much less sensitive to noise than the conventional Laplace or Sobel filters.

Actually, the SUSAN filter does not explicitly compute the region by filling, but simply adds a similarity measure applied to all pixels in the kernel. Given a centre pixel $\vec{r}_0$ this similarity is defined as

$$c(\vec{r}, \vec{r}_0) \quad = \quad e^{-\left(\frac{I(\vec{r})-I(\vec{r}_0)}{t}\right)^6}$$

where $I(\vec{r})$ is the value of a pixel $\vec{r}$ and t is a scaling parameter. This parameter determines the sensitivity of the filter as well as its dependence on noise: Smaller values allow less variation and larger values allow more variation. Using the sixth power is according to Smith a good

[14] See [Smith and Brady 1995] or [Berger 2000] for a summary.

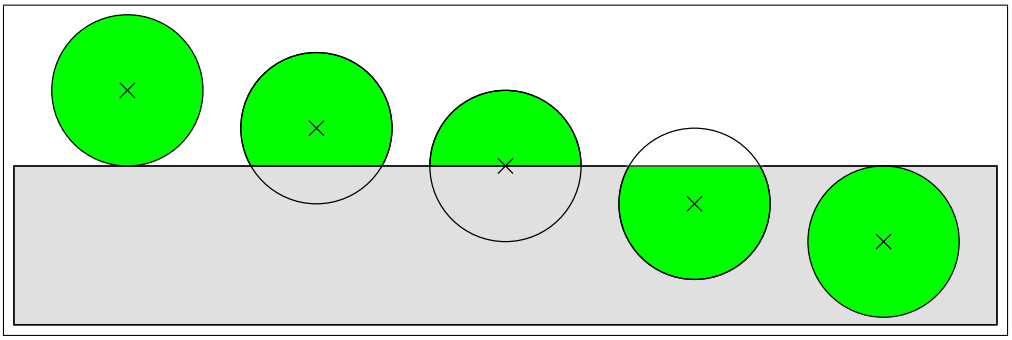


Figure C.18: The working principle of the SUSAN edge detector.

compromise between a relaxed sensitivity to the parameter t and a good transition from "good" to "bad" similarity. The (inverse) strength of an edge is then computed as the "size"

$$n(\vec{r}_o) \quad = \quad \sum_{\vec{r}} c(\vec{r}, \vec{r}_0)$$

of the region of similar pixels over all pixels $\vec{x}$ in the kernel around $\vec{x}_0$. Besides using this strength the direction of the edge can also be computed to allow non-maximum suppression, but for brevity this is dropped here.

Figure C.19 shows some example for the SUSAN filter with different kernel sizes and similarity thresholds. The images have been inverted (so that white corresponds to an edge) and contrast-maximised like the images for the other edge detectors. Compared to the Laplace and Sobel filters the edges show up more clearly, although parameter tuning is – as always – necessary. The number of levels of edge strength (and gray levels in the images) is linked to the kernel size: Smaller kernels do not allow for a large number of levels of edge strength. To allow sufficient differentiation between edge strengths the kernel must therefore not be too small. Disturbances in the output resulting from noise in the input can be effectively suppressed by increasing the threshold, which allows larger areas to be found similar. With the other filters a-priori smoothing would have to be applied to remove noise, and this would in turn lead to that edges are blurred too. With the SUSAN filter this problem does not exist.

C.2.5 Segmentation

Another important manipulation procedure is image segmentation. It means to find regions of contiguous pixels that satisfy some criterion. By describing larger entities rather than single pixels it reduces the information dimensionality significantly. It can be used to extract relevant information in a pre-processing step to recognise complex objects. There are two basic ways to look at segmentation.

C.2.5.1 Top-Down Approaches

One way to implement segmentation are *top-down* approaches which start with the image as a whole and split it in quad-trees as long as the criterion is violated. This process is repeated recursively until for all those parts where the criterion is still violated. Since – as can be seen in figure C.20 – the splitting is only done at fixed points it has difficulties in reproducing the shape

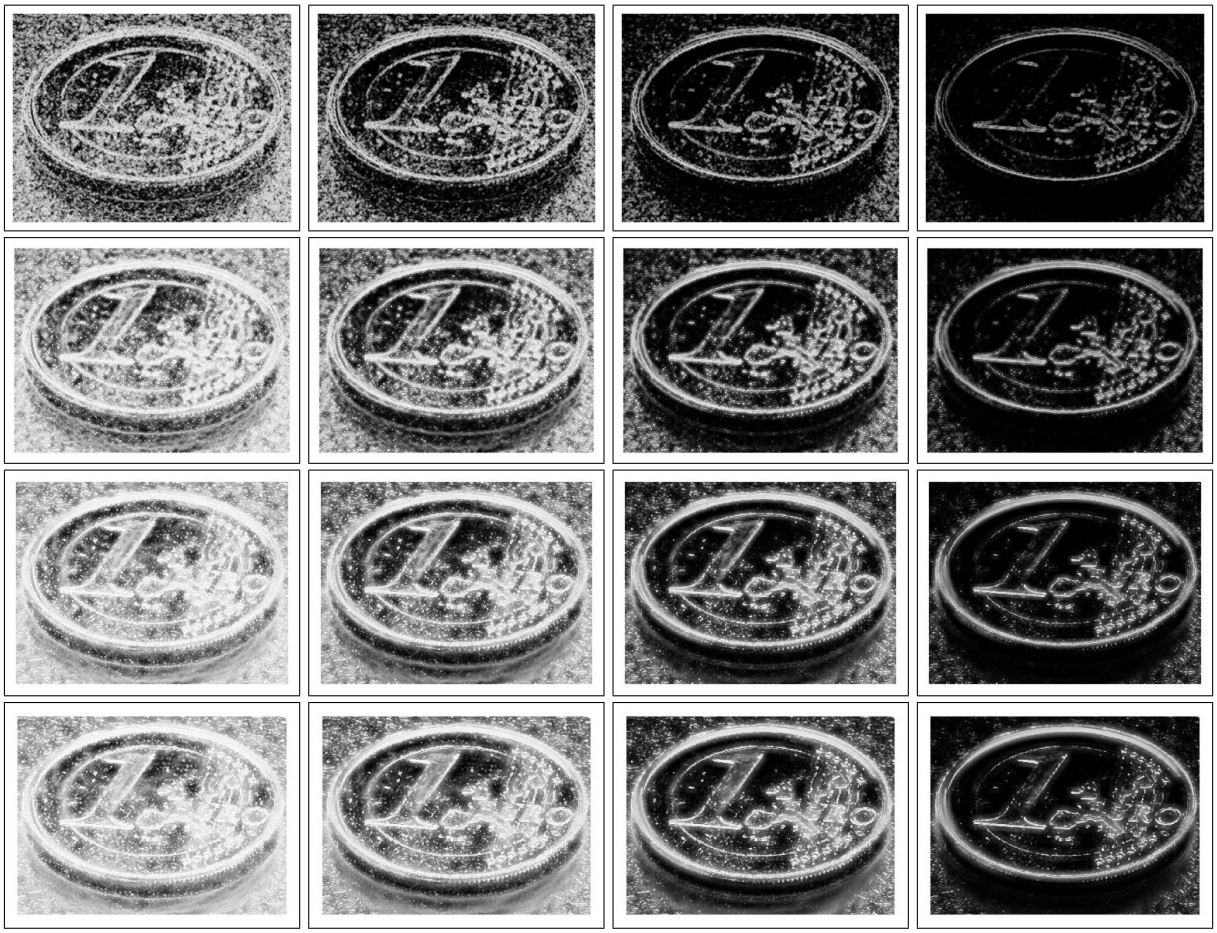


Figure C.19: Outputs from the SUSAN edge detector. Rows from top to bottom: kernel size of 3×3, 7×7, 11×11 and 15×15. Columns from left to right: scale values of 4, 8, 16 and 32.

of objects. For a good object shape reproduction the minimum cell size and similarity threshold must be very small and a second processing step added, in which adjacent regions are merged again if their union does not violate the constraint again. This technique is therefore also called *split & merge*.

One example of such a top-down technique is the Linde-Buzo-Gray (LBG) algorithm, a modified k-means algorithm[15]. The normal k-means algorithm uses a-priori knowledge about the number of clusters to segment the input space into, for example, speaking in terms of colour image segmentation, the number of independent colours. Having to know the number of colours in advance is of course a severe disadvantage because it cannot be guaranteed that their number is sufficient. On the other hand it may perhaps be too high for some images, and using too many colours means wasting information and complicating the later segmentation. It is this second point which the LBG algorithm addresses. It starts with one cluster and recursively splits clusters until a maximum number of clusters is hit or the quantisation error is below some threshold.

Such a colour quantisation is but one step of the image segmentation. It reduces the colour information to a small number of colour indices, but it depends on the number of possible colour indices whether they represent the eventually desired regions. Since in order to avoid problems

[15] See [Röben 2003].

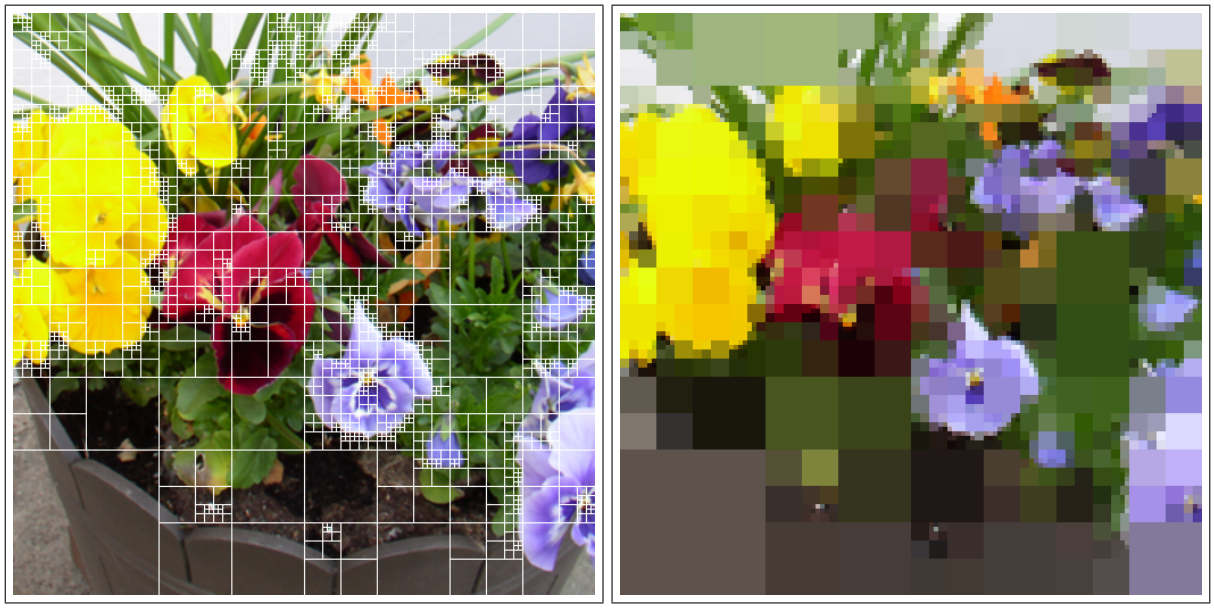


Figure C.20: Image segmentation by quadtree splitting. Starting with the entire image, each region is split in four quarters if a similarity criterion is violated. The image on the left shows the original image with the quadtree regions and the image on the right shows the regions filled with their average colour.

that limit is usually deliberately set too high than too low an additional merge step may be necessary to join adjacent regions of similar colour.

C.2.5.2 Bottom-Up Approaches

Another way to achieve a segmentation is to use a *bottom-up* approach that works the other way round. Instead of beginning with the complete image and splitting regions as long as they violate some constraint the bottom-up approach starts at a single pixel and tries to fill a region around it as long as pixels satisfy the constraint. This is also called *region growing*. As can be seen in figure C.21 contrary to quadtree splits this approach can very precisely represent the shape of objects.

Region growing approaches do not need any knowledge about the number of clusters to segment its input into, but can in the worst case yield one cluster per input data if the criterion is too sharp. Another potential problem with region growing often mentioned in literature is the selection of the initial pixel: If the *seed* is not chosen optimal the algorithm may produce more clusters than necessary. In term of colour segmentation this problem does not occur because the colour saturation can be used as a means to determine seed pixels. It is because of this interesting feature that a region-growing approach has been chosen for the colour segmentation in this work.

C.2.6 Feature Based Classification

Classification is the task to recognise objects in an image. Since in general more than just one object has to be recognised it is usually formally described as the task to assign an image $\vec{b}$ to

Figure C.21: Image segmentation by region growing. Starting with an initial pixel, regions are build by a flood fill algorithm as long as a similarity criterion holds. The image on the left shows the original image and the image on the right the regions filled with the colour of the initial pixel.

one of a set of N possible classes.

$$\vec{b} \quad \rightarrow \quad i \in [1..N]$$

As such, it only covers the aspect of recognising the object and not that of determining its position in the image. It also means that most classification algorithms only allow to recognise *known* situations. Unless they have an explicit rejection class an unknown situation will be taken as one of the known ones, although with a high error. There are techniques for dealing with this and adding a new class when the classification error gets too high, but they are not presented here because due to the fixed number of objects used in this work they are not of interest.

What will be presented is an overview over some classification techniques helpful for understanding some inherent problems of classification. Since the classification implemented for this work uses different techniques this overview is only very brief[16].

The basic idea of all these classificators is to place a hyper-curve into the input space that allows to tell whether a specific input pattern is on one side of the plane or the other. This means that classification is not achieved by storing information about one class only, but by explicitly comparing two classes. As a consequence it is not sufficient to have a set of sample images of an object to be recognised. In addition to these positive examples a set of negative examples is also needed to give the classificator the possibility to compute the "border" that allows to distinguish them. Therefore it is strictly speaking not possible to recognise a single situation, but only to distinguish at least two situations.

The techniques shown in this subsection are often called *feature based* classification techniques. Although defined on abstract vectors, they are usually not applied to complete images,

[16]For a real introduction into classification and/or pattern recognition see for example [Niemann 1983].

but only to reduced information computed from them – the features. One reason for doing this is that these techniques can need a very high number of parameters to approximate a separating function with enough accuracy to allow to distinguish all possible images, making them slow to evaluate and uncomfortable to handle. In some cases the storage space required for the parameters can even exceed the memory capacity of contemporary computers, in particular with the B-Spline Fuzzy Controllers[17].

In addition to this, the more parameters a system has the more degrees of freedom it has to adjust itself, which may cause undesired side effects. Interpolation of a function given by a large number of sample points, for example, is not done by fitting a single high dimensional polynomial to the data, but by piecewise fitting several low-dimensional polynomials. The single high-dimensional polynomial has too many degrees of freedom that allow it to match the sample points perfectly and yet wildly oscillate in between. Features extraction is therefore an important step in reducing the dimensionality of the problem and making the classification more reliable.

C.2.6.1 Neural Networks

One well-known modern classification technique are *neural networks* (NNs). NNs are inspired by nature, which uses an electric system to transmit information over neural tracts. This information is not encoded in the voltage, but in the frequency of occurrence of impulses. The neurons are cells that receive electric impulses from other neurons and, depending on the frequency of occurrence of these impulses, themselves send out impulses to again other neurons.

In a computer simulation this situation has to be simplified (modelled) because computers cannot handle frequencies well. Information is therefore again encoded as the magnitude of a signal, not the frequency of its occurrence. A computer neuron then is an entity that receives a vector $\vec{x}$ of input from other neurons, performs a scalar product with a vector $\vec{w}$ of weights to obtain a weighted sum of that input and finally outputs a signal

$$y \quad = \quad f_{\text{activate}}(\vec{x} \cdot \vec{w})$$

according to an activation function $f_{\text{activate}}()$. This activation function is what makes artificial NNs interesting and powerful tools. Without the activation function (or actually: with any *linear* activation function, including $f(x) = x$) the NN would compute only a linear combination of its input and the weights. Such a linear combination describes a hyper-plane in a high-dimensional space, which – in terms of classification – means that only linear separation is possible. The NN would be less powerful than polynom classificators.

In addition to this, NNs have to be *trained*, which means they define an error function on their output which is to be minimised iteratively. For the majority of applications this is done by the *backpropagation* algorithm which uses partial derivatives of the output of each neuron to change the weights for its inputs. However, using derivation of a linear function is somewhat pointless and unlikely to yield a stable converging of the set of weights. As a result other, non-linear activation functions are usually used, mostly one of the class of sigmoid functions like the tangens hyperbolicus in figure C.22. These functions allow a non-linear combination of the inputs as well as stable converging because they saturate at certain values and their derivation becomes zero.

[17] This problem is also known as the *curse of dimensionality*.

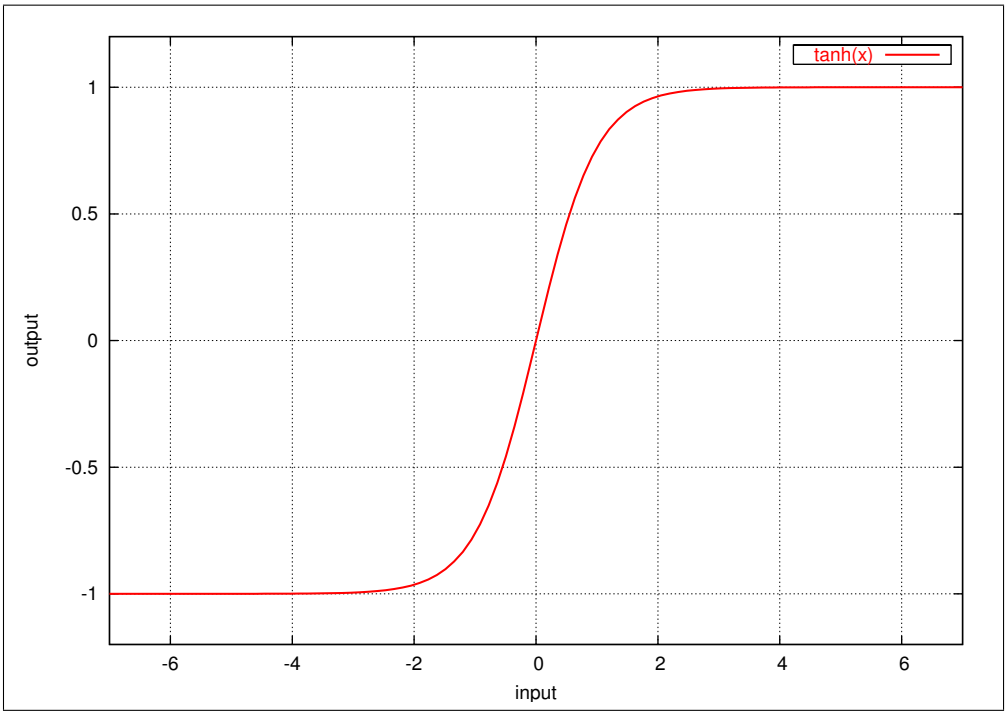


Figure C.22: The tangens hyperbolicus function as a neural activation function.

Such a single neuron is called a perceptron and is not yet very powerful. In fact, Minsky and Papert have shown that it can not even compute the XOR problem[18]. The potential that arises from perceptrons comes from joining several ones into a *multi layer perceptron* (MLP) as in figure C.23. Such a MLP consists on an input layer, one or more hidden layers[19] and an output layer. The number of neurons in each layer varies according to the task the MLP has to fulfil. With these MLP arbitrary complex problems can be computed.

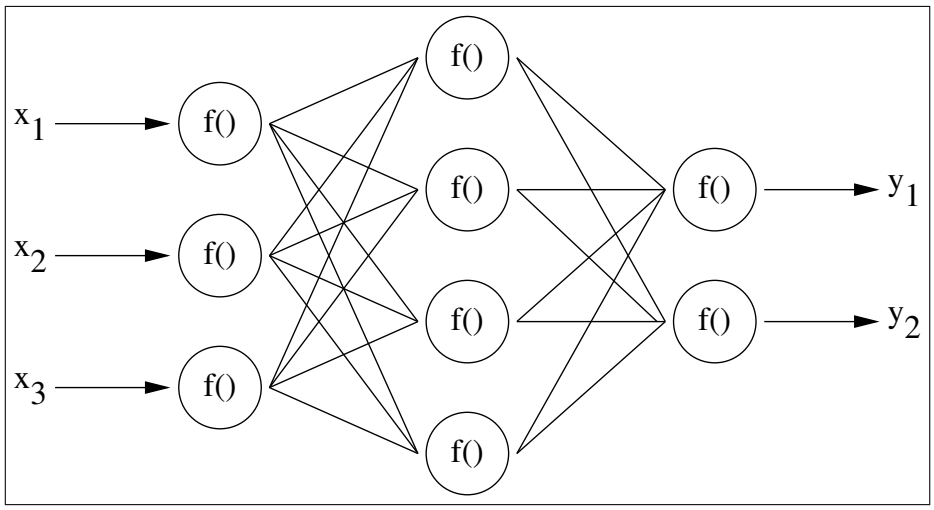


Figure C.23: A sample multi-layer-perceptron.

[18] See [Minsky and Papert 1969].

[19] It can be proven that one layer of hidden neurons is functionally complete, but often practical reasons speak for using more layers.

Such a MLP/NN itself does strictly speaking not yet classify, but only learns to approximate an arbitrary function

$$y = f(\vec{x})$$

from its input pattern. Classification capabilities can be achieved by teaching an output of 1 in case of an input pattern that belongs to a class and 0 for all other patterns[20]. After training the network will then compute an output that is unlikely to be exactly one of these desired values, but is expected to be very close to one of them. Depending on which of the desired values it comes closer to the classification can be answered with yes or no.

Since the net learns a *continuous* function classification into multiple classes can theoretically be done by learning several distinct values for each class of input patterns. This, however, remains a theoretical approach because it is likely to fail due to an inaccurate approximation of the function. A much more stable approach is to use several output neurons – one for each class of input patterns – and learn a set of functions

$$\vec{y} = f(\vec{x})$$

instead of just a single one. For each input pattern the output neuron to whose class the pattern belongs is given 1 as desired output and all other neurons 0. The output of the network after training will be an arbitrary vector in which one component is expected to be significantly larger than the other ones. The index number of this component gives the MLP's belief to which class the input pattern belongs.

C.2.6.2 Fuzzy Controllers

Another approach to classification are fuzzy techniques like the B-Spline Fuzzy Controllers (BSFC)[21]. The BSFC places a hyper-grid of b-spline functions of a certain degree over the input space. These functions are designed so that they react to input only in a limited local range and that a couple of other conditions are met (for example the *partition of unity*). By assigning each function an individual weight the input data function can then be approximated by computing

$$y = \sum_{i_1} ... \sum_{i_k} w_{i_1,...,i_k} \cdot f_{i_1,...,i_k}(\vec{x})$$

as output of the BSFC. The weights are adjusted iteratively by a simple learning algorithm. Since the error function in this algorithm is quadratic it has only one minimum and the algorithm can therefore be proven to converge.

The BSFC is similar to a NN, especially to the group of *radial basis function* (RBF) NNs – they all are generic function approximators. As such, it also learns a multi-dimensional function that allows to separate different classes of input patterns. Because of the need to store a hyper-grid of weights that has as many dimensions as the input vector has elements BSFCs can only be used for low-dimensional problem. It is completely impossible to use them on images directly.

[20] More precisely, for symmetric functions like sigmoid functions it is recommendable to learn symmetric values, for example in case of $\tanh()$ the extrema -1 and +1 at which it saturates. On the other hand, the backpropagation algorithm uses the derivation of the function and will converge only very slowly at points where that function is almost flat. Therefore, even more precisely, in case of $\tanh()$ it is advisable to learn values a little bit smaller, like -0.9 and +0.9.

[21] See [Zhang and Knoll 1996] and [Zhang and Knoll 1998].

C.2.7 Appearance Based Classification

The classification techniques presented above have been researched for a long time. Their advantages and disadvantages are therefore well known. One of their biggest disadvantage is that the computation of the features they use is not necessarily trivial.

For example, one can try and classify a certain group of images by checking whether there is "a lot of blue" in the upper part of the image and "a lot of green" in the lower part. Apparently, this will hold true for a lot of scenic landscape images, so the required features are rather simple in this case. Telling how many mountains there are in the image is a lot more difficult because it requires much more complex features – something like analysing the horizon and looking for peaks.

This disadvantage of classic approaches has led to an increasing desire to use techniques which enable a computer to perform the classification by its own and without prior computation of complex features. With no a-priori knowledge about the situations built-in to the algorithms these techniques are by definition all learning techniques. In this subsection two of these techniques will be shown.

Listing them under appearance based classification does not mean that they cannot be used for feature based classification, it just means that they primarily focus on appearance based classification because they focus on high-dimensional data. Their central idea is to use a high-dimensional off-line computation based on learn data that allows to reduce the dimensionality of the work data on-line. They both compute

$$\vec{x}_{\mathrm{n}} \;=\; M \cdot \vec{x}_{\mathrm{o}}$$

where M is a matrix and $\dim(\vec{x}_{\mathrm{n}}) \ll \dim(\vec{x}_{\mathrm{o}})$ while $\vec{x}_{\mathrm{n}}$ is supposed to still hold as much information as possible.

Again the overview will only briefly mention the most important aspects because these techniques have been found to be not applicable to this work. The inherent problem is that due to uncertain generalisation an explicit sample image has to be learned for any situation that has to be recognised, and this is not feasible in case of many objects used in this work. For example the cage in the centrifuge can legally be rotated by 360 degrees, but it is not possible to take all corresponding learn images automatically with the robot because for the larger rotations it will then collide with the lid of the centrifuge. Even in cases where using the arm to generate rotations does not lead to collision with objects it can still be impossible because it would drive the intended target out of range. Generating the images manually is not an alternative because it would require very complex auxiliary structures to ensure the necessary accuracy.

On the other hand, not including the training data that cannot be generated automatically does not simply mean that the algorithm will recognise this and/or compute no result, but will instead lead to severe errors. Both the presented approaches will in this case return an output that belongs to a learned input which – in their internal representation – is as good as "similar" to the real input as possible. As such, it will be wrong and can lead to severe damage to the device and/or the robot. From the two solutions to find other ways of generating learn images or to use a completely different approach the second one has been chosen for this work.

C.2.7.1 Principal Component Analysis

The *principal component analysis* (PCA) is a well-known linear transformation of vector spaces used for dimensionality reduction. It is based of the fact that every linear transformation of a vector space is equivalent to a change of the set of base vectors $\vec{b}_i$ spanning the vector space. A linear transformation of m-dimensional vectors $\vec{x}^{(m)}$ can therefore be written as

$$\vec{y}^{(m)} = B \cdot \vec{x}^{(m)}$$

with $B \in \mathbf{R}^{m \times m}$ being the matrix of new base vectors. Obviously, the dimensionality of the vector space can be reduced by using only $n < m$ base vectors and compute

$$\vec{y}^{(n)} = B' \cdot \vec{x}^{(m)}$$

with $B' \in \mathrm{R}^{n \times m}$. However, this discards information from the $\vec{x}^{(m)}$. Speaking about separation, it is of course easier (needs less accuracy and thus less parameters) to separate data that is spread over a wider range, as can be seen in figure C.24. So, in order to maximise the separability a minimal set of new base vectors has to be chosen that maximises the data entropy.

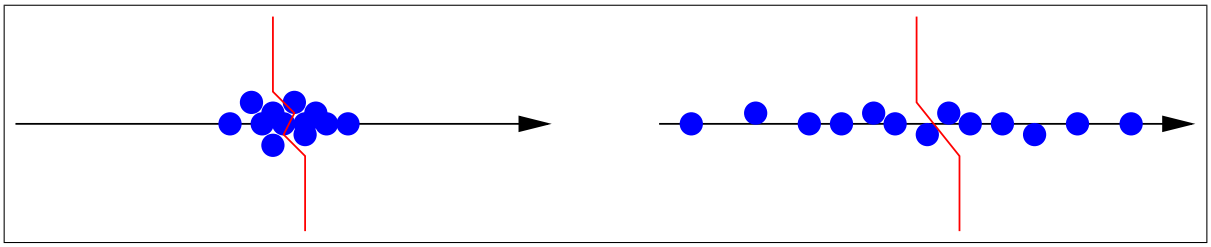


Figure C.24: Two (mostly) 1-dimensional distributions visualising the separability problem.

At this point an interesting feature of linear vector space transformations can be used: Every linear vector space transformation M has a set of vectors $\vec{e}$ for which

$$\lambda_j \cdot \vec{e}_j = M \cdot \vec{e}_j$$

holds, meaning that these vectors are only stretched, but not rotated. These vectors are called *eigenvectors* and their associated parameters λ_j are called *eigenvalues*. If a data distribution uses all m dimensions of its vector space it can be shown that m distinct eigenvectors with eigenvalues $\neq 0$ exist. It the distribution does not use all dimensions some eigenvalues will be 0 and the corresponding eigenvectors will not exist. Figure C.25 gives an example of the eigenvectors for a 2-dimensional distribution.

The eigenvalues λ_i represent the data variance along the eigenvectors. When using a subset of $n < m$ of these eigenvectors as a new vector base system for dimension reduction it can therefore be shown that the error e_n describing the loss of information by dropping dimensions is proportional to the sum of the eigenvalues of the unused eigenvectors.

$$e_\mathrm{n} \sim \sum_{i=n+1}^{m} \lambda_i$$

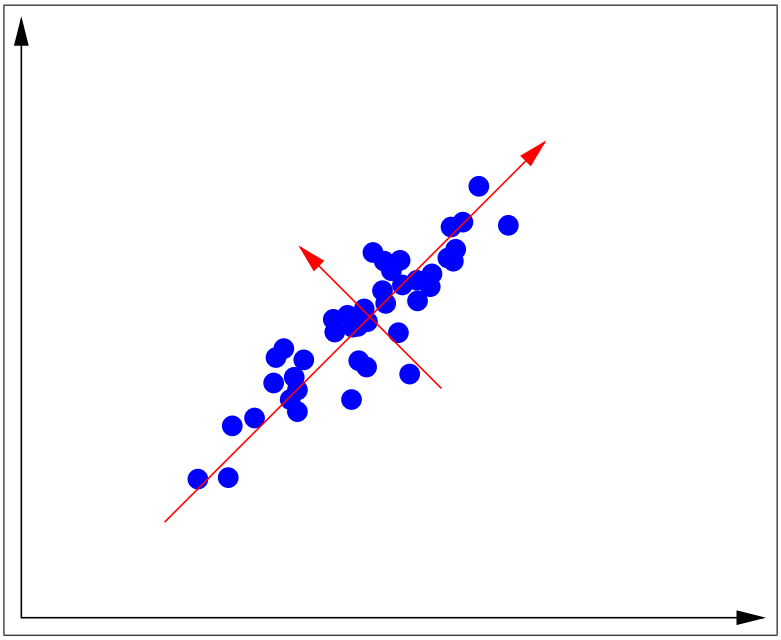


Figure C.25: A 2-dimensional distribution with its eigenvectors.

This error can obviously be minimised by sorting the eigenvalues and using those eigenvectors that belong to the n largest eigenvalues.

Based on this principle, the PCA takes a set of sample data $\{\vec{x}\}$ and computes the mean (expected value)

$$\vec{\mu} \quad = \quad E(\vec{x})$$

and the covariance matrix

$$C \quad = \quad E\left((\vec{x} - \vec{\mu})(\vec{x} - \vec{\mu})^T\right)$$

of the data distribution[22]. It then decomposes the covariance matrix C into its eigenvectors, selects a subset of these vectors and build a matrix $B' \in \mathbf{R}^{n \times m}$ from them. This is done off-line on learning data. At run-time the dimensionality of an input vector $\vec{x}^{(m)}$ can then be reduced by applying

$$\vec{y}^{(n)} \quad = \quad B' \cdot \left(\vec{x}^{(M)} - \vec{\mu}\right)$$

while maintaining the maximum entropy.

The PCA appears to be a very promising approach to reduce the dimensionality of a problem, but it does have several drawbacks. The biggest drawback is that it is an *unsupervised* and stochastic approach. It tries to reduce the dimensionality while maintaining the general data entropy, silently assuming that this is the information the user is interested in. This, however, is not necessarily true.

The PCA treats data as a distribution, blindly looking at variations of the values. In case of images there are a lot of variations possible which the user may not be interested in, like

[22] Actually, the covariance matrix can be too large to be handled by a contemporary computer, in which case the *implicit covariance matrix* is used instead. This is a simplification which has no effect on the results.

changes in brightness and colour, image noise and minor changes in appearance of the objects due to lens aberrations or (partial) occlusion. If such a variation would dominate the more relevant variations (lead to a higher eigenvalue) it will be regarded as more important and the eigenvectors will therefore focus on something "wrong". As an absolutely essential prerequisite the images have to be normalised in brightness and size to ease this problem, but even then it is not completely eliminated. Under these circumstances the fact that the PCA also requires a comparatively high amount of memory for parameter storage has to be seen as only a minor handicap.

The PCA has initially been evaluated for this work by C. Berger[23]. At the time of his analysis the intention was only to use the PCA to recognise low-level geometric shapes like circles, rectangles and triangles, but not to detect their position in the image. The objects were extracted from the image using early versions of the colour image segmentation presented in this work, and normalised in size and brightness. The main variance that the PCA was intended to focus on was that cause by object orientation, which was for obvious reasons not normalised. The resulting set of training images of the rotated objects was used to compute a PCA.

One aspect of "maintaining information" during a PCA is that images which are similar are reduced to similar vectors in the eigenspace. As can be seen in figure C.26 the different orientations of the objects lead to contiguous trajectories in the eigenspace. The vectors for the "compressed" images are stored and used to classify objects at run-time using a next-neighbour test.

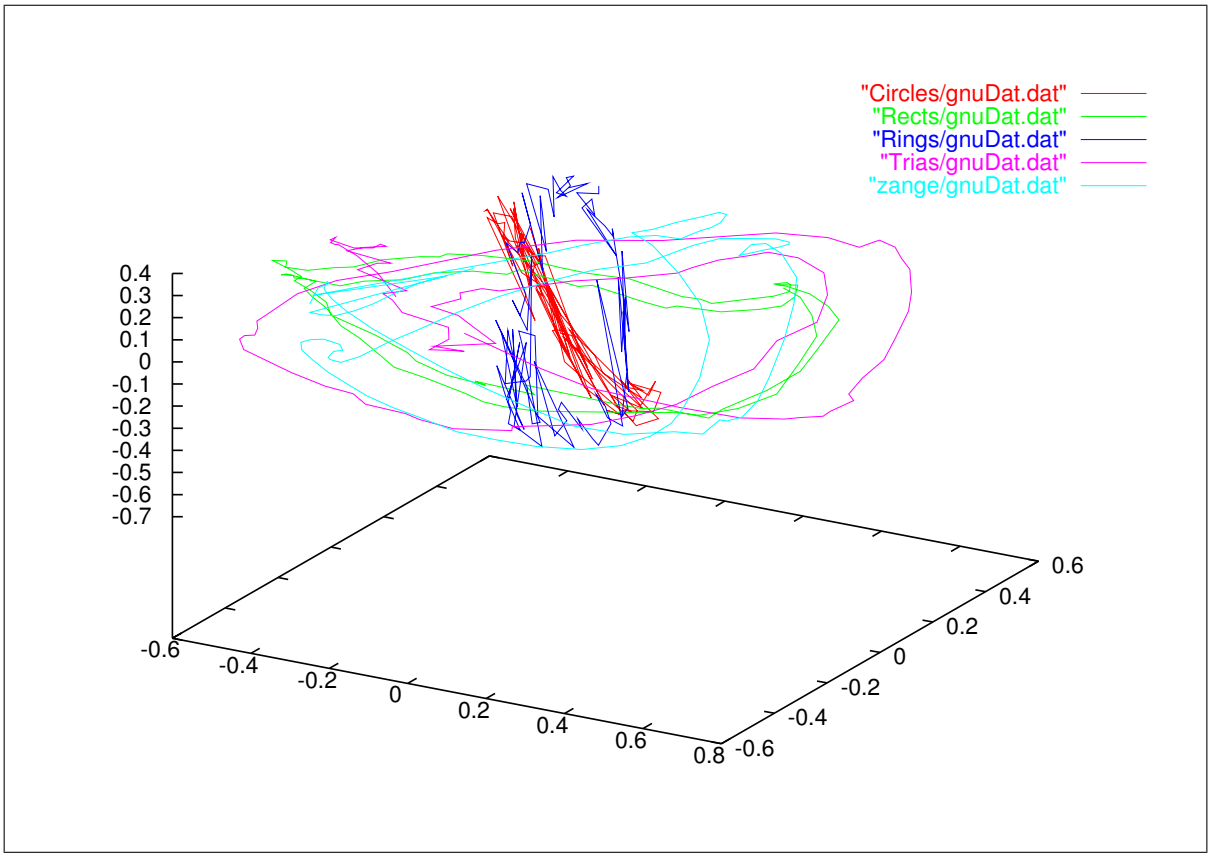


Figure C.26: Visualisation of the global eigenspace of five different geometric objects.

[23] See [Berger 2000].

Berger tested several variants of the PCA[24]: One with 100 learn images and 10 eigenvectors and one with 360 learn images (1 image per degree of object rotation) and 5, 7, and 10 eigenvectors. His classification results – compared to a geometric approach for which no normalisation is needed – can be seen in table C.3. Berger has shown that a classification of simple objects using PCA is possible, although the number of dimensions to use remains a critical parameter.

	#images	#dim	triangle	circle	rectangle	ring	tongs
geom	-	-	84	94	90	82	-
PCA	100	10	100	100	100	98	100
PCA	360	5	74	44	78	78	100
PCA	360	7	92	58	92	92	100
PCA	360	10	100	100	100	100	100

Table C.3: Classification results of several PCA variants. All numbers are probabilities for correct-positive classifications in percent.

Since the laboratory scenario consists of more complex objects than simple geometric forms Berger's approach had to be extended. Tests have been made in which objects consisting of several regions were to be classified. These tests have shown the problem that with no a-priori knowledge about what object is in the image and where in the image it is the object's regions cannot be separated from the background. This means that the normalisation of the size required for the PCA is not possible.

Attempts to not segment the object from the background and not normalise its size, but use the complete image instead have not nearly been as successful as the results for the simple geometric shapes. The variations in the images introduced by allowing the objects to translate were too big and virtually suppressed the variations arising from the different objects. Using the PCA on complex objects consisting of several non-contiguous regions that cannot be separated from the background in advance has therefore to be considered unfeasible.

C.2.7.2 Output Related Features

As has been shown the PCA is an unsupervised approach that uses generic principles from information theory. A more specific approach following similar ideas are *output related features* (ORFs). With ORFs the reduced set of new base vectors $\vec{b}$ is not automatically computed by using the data variance, but by trying to maximise linear correlation

$$\vec{b} \cdot \vec{x}_i \quad = \quad s_i$$

to a user-supplied desired output s_i for each input $\vec{x}_i$. As such, ORFs implement an approach based on *supervised* learning, the learning rule of which is

$$\Delta\vec{b} \quad = \quad \epsilon \cdot \vec{x}_i \cdot \left(s_i - \vec{x}_i \cdot \vec{b} \right)$$

with some $\epsilon \ll 1$ used as scaling factor. Once one ORF vector $\vec{b}$ is known the information along that vector can be taken from the data by computing

[24] See [Berger 2000], chapter 5, page 40ff.

$$\vec{x}_i' \quad = \quad \vec{x}_i - \vec{b}\left(\vec{b} \cdot \vec{x}_i\right)$$

for all data vectors $\vec{x}_i$ and repeating the learning process to obtain another ORF vector.

Since the user-supplied desired output is completely arbitrary, ORFs can be used for many tasks. If, for example, the desired output is an index number they can be used for classification. If the desired output is the coordinate of an object in the image this can also be learned – at least theoretically. The ORF approach has therefore been tested on learning the coordinates of objects in images[25]. Unfortunately its capabilities to interpolate between the learned grid data have been found to not meet the required accuracy. In addition to this, also suffers from the extrapolation problem occurring when not all possible situations have been trained.

The most important disadvantage of the ORF approach is its linearity. Maximising linear correlation with the desired output implies that there is a linear correlation between input and output, which is not necessarily true. An example of the possible problems can be seen in figure C.27 where the input data is spread on a semi-arc. When training ORF vectors to classify the data it is reasonably easy to obtain one vector, although the correlation is apparently not really linear. One way to avoid classification problems in the non-linear areas would be to use a second vector, hoping that this one would perform better. Projecting the data onto the subspace, however, leads to a a distribution where not even an roughly approximated linear correlation is possible. The second vector will be more or less random and will not improve the classification.

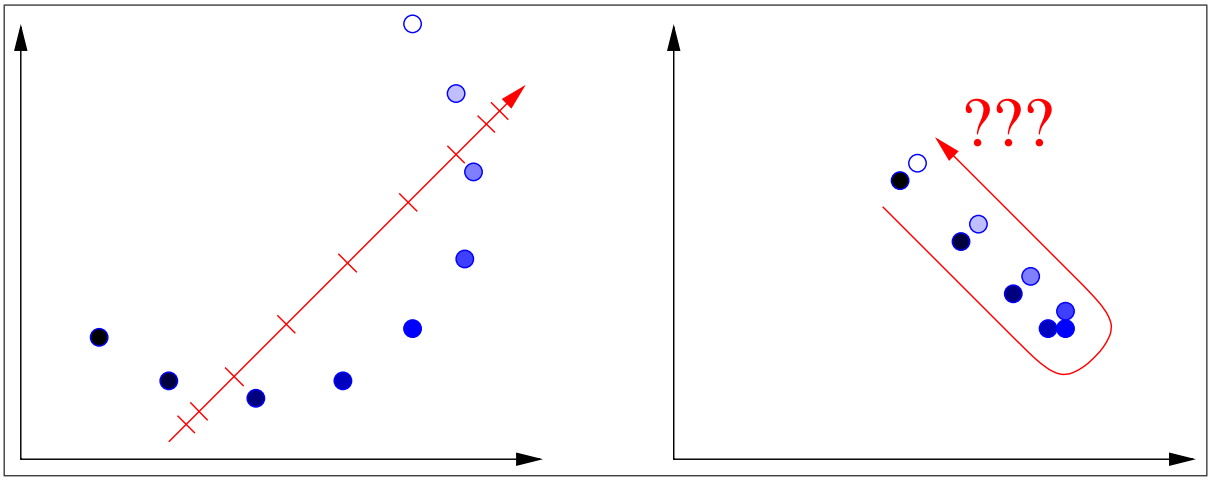


Figure C.27: A sample data distribution demonstrating problems of the ORF approach.

Using ORFs is only feasible in cases where the correlation between the input and the desired output *is* linear, and in high dimensional spaces like images this is difficult to say in advance. In case of the the recognition of the class and position of the objects used in this work it has shown to be not the case. ORFs can therefore not be used.

C.2.8 Summary

The history of computer vision has produced a lot of techniques to operate on images, however, some of them are pretty theoretical and of limited practical use. For example the Sobel filter

[25] To not make things too difficult only the simple geometric forms used in the PCA tests have been used.

is the “correct” way to detect edges, because edges are derivations of the value and it computes the derivation to detect them. Unfortunately it also “detects” image noise, and as a result of this the plain Sobel filter is very complicated to use. Other filters that use more complex computations to avoid these problems like the SUSAN filter are still not of widespread use, but are urgently needed by a lot of practical applications. The manipulation techniques used in this work are therefore almost entirely somewhat unconventional, but strongly adapted to the specific problem.

Concerning classification it has to be mentioned that none of the presented approaches “recognises” an *object* the way a human does, but only a very reduced set of more or less abstract *features* of the object – be that because of explicit feature extraction or because of a representation in an eigenspace. One reason for this is that it is still not yet known how the human vision system works. It appears certain that the human brain also reduces a colour image as delivered by the eye to abstract features like edges, corners and areas, but neither the set of features nor the exact type of the classificator are known.

With the current level of knowledge about the human vision system the artificial classificators can only be very coarse approximations. In particular, the PCA and ORF are linear approaches – a popular simplification – but the problem is obviously inherently non-linear. With such radical simplifications it is no wonder that their results reach only a fraction of the performance of the human eye. Computer vision system are usually called “good” if they yield a correct classification in more than 95% and/or if they compute positions of objects within a range of a few, but still several pixel. For an automated laboratory robot this would basically mean that it would not save any money on personnel because a human operator would have to constantly supervise it. Anything less than 100% performance is therefore not tolerable, and for such performance ratings other techniques have to be used.

Another aspect is that all the presented classification approaches need a set of learn images covering the *entire* range of possible situations. With some objects used in this work it is not possible to automatically obtain such a set of images, and so the classificators are forced to extrapolate. This, however, none of them can do in any useful way. It is primarily this reason why for this work a different approach has been chosen.

Appendix D

C++ Class Documentation

D.1 CINFOBASE – Accessing the INFOBASE

The C++ class `CINFOBASE` defined in the header file `infobase.h` implements the client access to the INFOBASE. It declares methods to access the server that are grouped in primary and secondary methods. The primary methods are declared as in table D.1 and explained as follows:

```
class CINFOBASE
{
public:
  CSTATUS Chdir (char *pName);
  CSTATUS Mkdir (char *pName, int flags = 0);
  CSTATUS Creat (char *fName, int flags = 0);
  CSTATUS Read (char *fName, void *buf, int bufMaxLength);
  CSTATUS Write (char *fName, void *buf, int bufLength);
  CSTATUS Stat (char *eName, int *isDir = 0,int *size = 0, double *time = 0);
  CSTATUS Unlink (char *eName);

  CINFOBASEDIR *Opendir (char *pName, int flags = 0);
};

class CINFOBASEDIR
{
public:
  CSTATUS Readdir (int *isDir = 0, int *size = 0, double *time = 0);
};
```

Table D.1: A (simplified) declaration of the primary methods of the `CINFOBASE` class.

CSTATUS CINFOBASE::Chdir (char *pName) sets `pName` as the new current working directory (path). With relative addressing (i.e. paths without a leading "`/`") this path is used as prefix.

CSTATUS CINFOBASE::Mkdir (char *pName, int flags) creates a new directory `pName`[1]. No entry (directory or file) with the given name must so far exist.

CSTATUS CINFOBASE::Creat (char *fName, int flags) creates a new file `fName`. No entry (directory or file) with the given name must so far exist.

[1]The `flags` parameter used for `Mkdir()` and `Creat()` is not used (yet). It is used in `Opendir()` though.

CSTATUS CINFOBASE::Read (char *fName, void *buf, int bufMaxLength) reads `bufMaxLength` bytes of data from the file `fName` into the buffer `buf`. If the file is smaller than the buffer only the file size is used. If the file is larger than the buffer only the prefix is used. In both cases the actual number of bytes read is returned. Data is treated as binary, which means that strings are explicitly not terminated by a trailing zero. Note that this way it is possible to read only a fraction of the file, but this fraction always starts at the beginning. It is not possible to read a fraction from the middle of the file.

CSTATUS CINFOBASE::Write (char *fName, void *buf, int bufLength) writes `bufMaxLength` bytes of data from the buffer `buf` to the file `fName`, discarding its previous contents. There is no limit for the data size except for the available memory.

CSTATUS CINFOBASE::Stat (char *eName, int *isDir, int *size, double *time) returns attributes of the entry `eName` (directory or file). The attributes are: The boolean flag `isDir` telling whether the entry is a directory or not and the integer `size` giving the entry's size in bytes (for directories: the number of its entries). The attribute `time` is not (yet) used.

CSTATUS CINFOBASE::Unlink (char *eName) deletes the entry `eName` (directory or file). In case of deleting a directory all its contents is automatically deleted too.

CINFOBASEDIR *CINFOBASE::Opendir (char *pName, int flags) opens the directory `pName` and returns a handle for it.

CSTATUS CINFOBASEDIR::Readdir (int *isDir, int *size, double *time) successively reads a directory, returning the next entry's name in `CSTATUS` and its attributes as in `Stat()`.

In addition to these primary methods `CINFOBASE` also declares a couple of secondary methods as shown in table D.2. These secondary methods are entirely based on the primary methods and only given for convenience reasons. Their explanation is as follows:

```
class CINFOBASE
{
public:
  CSTATUS ReadString (char *fName, char *value, int maxSize);
  CSTATUS ReadString (char *fName, char **value);

  CSTATUS ReadInt (char *fName, int *value, int maxSize = 1);
  CSTATUS ReadFloat (char *fName, float *value, int maxSize = 1);
  CSTATUS ReadDouble (char *fName, double *value, int maxSize = 1);

  CSTATUS WriteString (char *fName, char *value);

  CSTATUS WriteInt (char *fName, int value);
  CSTATUS WriteInt (char *fName, int *value, int size = 1);
  CSTATUS WriteFloat (char *fName, float value);
  CSTATUS WriteFloat (char *fName, float *value, int size = 1);
  CSTATUS WriteDouble (char *fName, double value);
  CSTATUS WriteDouble (char *fName, double *value, int size = 1);
};
```

Table D.2: A (simplified) declaration of the secondary methods of the `CINFOBASE` class.

CSTATUS CINFOBASE::ReadString (char *fName, char *value, int maxSize) reads the contents of a file `fName` into the C string `value`, whose size must be given in `maxSize`. If the file is smaller only a fraction of that string will be used, and if the file is larger only a fraction of the file will be read. In any case the string is terminated by a trailing zero, so the buffer size must take that into account.

CSTATUS CINFOBASE::ReadString (char *fName, char **value) reads the contents of a file `fName` into the C string `value` given by its pointer. The string is allocated by `malloc()` with the appropriate size to hold the data including a trailing zero. The previous value of the pointer is not evaluated. If it initially pointed to another string that memory reference is lost forever.

CSTATUS CINFOBASE::ReadInt (char *fName, int *value, int maxSize) reads a set of size `maxSize` of integers from the file `fName` to the array `value`. The array must be preallocated and large enough to hold the data. The data is read as an ASCII string and converted to integers using `strtol()`.

CSTATUS CINFOBASE::ReadFloat (char *fName, float *value, int maxSize) reads a set of size `maxSize` of floats from the file `fName` to the array `value`. The array must be preallocated and large enough to hold the data. The data is read as an ASCII string and converted to floats using `strtod()`.

CSTATUS CINFOBASE::ReadDouble (char *fName, double *value, int maxSize) reads a set of size `maxSize` of doubles from the file `fName` to the array `value`. The array must be preallocated and large enough to hold the data. The data is read as an ASCII string and converted to doubles using `strtod()`.

CSTATUS CINFOBASE::WriteString (char *fName, char *value) writes the string `value` to the file `fName`, deleting its previous contents. The file must exist to be written to.

CSTATUS CINFOBASE::WriteInt (char *fName, int value) writes a single integer `value` to the file `fName`, deleting its previous contents. The file must exist to be written to. The integer is converted using `printf()` with an `%i` argument and stored as an ASCII string.

CSTATUS CINFOBASE::WriteInt (char *fName, int *value, int size) writes the array `size` containing `size` integers to the file `fName`, deleting its previous contents. The file must exist to be written to. The integers are converted using `printf()` with an `%i` argument and stored as an ASCII string.

CSTATUS CINFOBASE::WriteFloat (char *fName, float value) writes a single float `value` to the file `fName`, deleting its previous contents. The file must exist to be written to. The float is converted using `printf()` with an `%.10e` argument and stored as an ASCII string.

CSTATUS CINFOBASE::WriteFloat (char *fName, float *value, int size) writes the array `size` containing `size` floats to the file `fName`, deleting its previous contents. The file must exist to be written to. The floats are converted using `printf()` with an `%.10e` argument and stored as an ASCII string.

CSTATUS CINFOBASE::WriteDouble (char *fName, double value) writes a single double `value` to the file `fName`, deleting its previous contents. The file must exist to be written to. The double is converted using `printf()` with an `%.20e` argument and stored as an ASCII string.

CSTATUS CINFOBASE::WriteDouble (char *fName, double *value, int size) writes the array `size` containing `size` doubles to the file `fName`, deleting its previous contents. The file must exist to be written to. The doubles are converted using `printf()` with an `%.20e` argument and stored as an ASCII string.

The return code `CSTATUS` used used these methods is also a C++ class as shown in table D.3. It implements a container for a pair of one integer and one string to be returned simultaneously. It has two cast operators `operator char* (void)` and `operator int (void)` to select the desired component. The string may either be a textual description of the error or user data like the next entry name in `Readdir()`.

```
class CSTATUS
{
constprotected:

  int _i;
  char *_s;

public:

  CSTATUS (void) { _i = 0; _s = 0x0; }
  CSTATUS (int i, char *s = 0x0);

  operator char* (void) { return _s ?  _s :  "(null)"; }
  operator int (void) { return _i; }
};
```

Table D.3: A (simplified) declaration of the `CSTATUS` return code class used to return both an integer error code as well as a string (printable error description or other content) at the same time.

Since the `int` cast is applied automatically by the C++ compiler for numerical comparisons this allows for a totally transparent usage like in:

```
if (infobase.Chdir ("/not-existent-directory") < 0)
{
  fprintf (stderr, "directory does not exist!\n");
  abort ();
}
```

If both values are needed the status can be stored and used as in:

```
CSTATUS status = directory->Readdir ();

if (status < 0)
{
  fprintf (stderr, "no more files\n");
  return;
}

printf ("%s\n", (char*)status);
```

D.2 CNETOBJ – Communicating over a TCP/IP network

D.2.1 CCHUNK

`CCHUNK` implements a (dynamic) buffer that the object is *serialised* into before being transmitted over the TCP/IP connection to the remote host. This buffer needs the same amount of memory as the object itself, which can become a problem for extremely large objects. On the other hand it is a simple solution to avoid a number of problems:

- Calling the system call `write()` for each individual data presents an overhead that is untolerable. With `CCHUNK` all data is written with a single `write()` call and the problem avoided.
- The TCP/IP stack of the operating system may buffer data before sending it, so if too few data is written it may actually not send it at all. With `CCHUNK` only large amounts of data are written and the problem avoided.
- In order to allow the receiver to skip unknown objects the size of the data to be trashed must be known. With no special encoding this requires the data size to be send before the data itself. Approaches like the ones used by Y. Collani and M. Ferch require an additional `GetSize()` recursion over the data before sending it to establish that size, which in turns reduces the encoding speed. With `CCHUNK` this second step is avoided at the price of an only slightly increased memory requirement that is certainly not a problem for the objects used in this work.

`CCHUNK` defines several `Pack()` methods to push data into the buffer and `Unpack()` methods to pop data from the buffer as in table D.4. Overruns are avoided by dynamically increasing the buffer size and underruns cannot occur[2]. The methods accept the commonly known `char`, `short`, `int`, `long`, `single` and `double` data types[3] used in C/C++, as well as `unsigned` versions of the integer ones and arrays. They properly encode the data so that hosts with different byte-ordering (big-endian vs. little-endian) can exchange data.

The most frequently needed pack/unpack methods are implemented directly:

[2]Since upon receiving an object its size is transmitted first and the buffer allocated and read accordingly no underrun can later occur when unpacking the object. If it does this means that the pack/unpack routines are not symmetric, which is a programming error and not one of the transmission framework.

[3]Currently, `char` is assumed to have 1 byte, `short` 2 bytes, `int` and `long` 4 bytes and `single` to be the 4 byte and `double` the 8 byte ANSI/IEEE-754 floating point values (see [IEEE 1985]). As far as the floating point formats are concerned this assumption is legal because virtually all contemporary architectures use them. The integer formats are a bit more troublesome, especially on 64-bit architectures. In case this should become a problem explicit `s8`/`u8`, `s16`/`u16` and `s32`/`u32` (and `s64`/`u64`) data types can easily be used instead.

```
class CCHUNK
{
public:

  CCHUNK (int chunkSize = 4096);
  CCHUNK (const void *data, const int size);

  virtual ~CCHUNK (void);

  int GetSize (void) const {return _written;}
  void *GetData (void) const {return _data;}

  // implement the (probably) most used methods directly...

  void PackInt (const int i);
  int UnpackInt (void);

  // ...all other are mapped to these by inline methods

  void Pack8BitArray (const void *ptr, int size);
  void Pack16BitArray (const void *ptr, int size);
  void Pack32BitArray (const void *ptr, int size);
  void Pack64BitArray (const void *ptr, int size);

  void Unpack8BitArray (void *ptr, int size);
  void Unpack16BitArray (void *ptr, int size);
  void Unpack32BitArray (void *ptr, int size);
  void Unpack64BitArray (void *ptr, int size);

  // all the following pack and unpack methods are based on the above

  void PackChar (const char c);
  void PackShort (const short s);
  void PackLong (const long l);
  void PackFloat (const float f);
  void PackDouble (const double d);

  void PackCharArray (const char *c, int size)
  void PackShortArray (const short *s, int size)
  void PackIntArray (const int *i, int size)
  void PackLongArray (const long *l, int size)
  void PackFloatArray (const float *f, int size)
  void PackDoubleArray (const double *d, int size)

  char UnpackChar (void);
  short UnpackShort (void);
  long UnpackLong (void);
  float UnpackFloat (void);
  double UnpackDouble (void);

  void UnpackCharArray (char *c, int size)
  void UnpackShortArray (short *s, int size)
  void UnpackIntArray (int *i, int size)
  void UnpackLongArray (long *l, int size)
  void UnpackFloatArray (float *f, int size)
  void UnpackDoubleArray (double *d, int size)
};
```

Table D.4: CCHUNK declaration. Methods with `unsigned` data types exist too, but are for brevity not listed here.

void CCHUNK::PackInt (const int i) pushes the 32 bit integer `i` into the buffer.

int CCHUNK::UnpackInt (void) pops a 32 bit integer off the buffer.

Except from the above, only the following array pack/unpack methods are implemented directly too. The pointers are of type `void *` because they are later used for multiple types.

void CCHUNK::Pack8BitArray (const void *ptr, int size) pushes the `size` elements from the 8 bit integer array `ptr` into the buffer.

void CCHUNK::Pack16BitArray (const void *ptr, int size) pushes the `size` elements from the 16 bit integer array `ptr` into the buffer.

void CCHUNK::Pack32BitArray (const void *ptr, int size) pushes the `size` elements from the 32 bit integer or float array `ptr` into the buffer.

void CCHUNK::Pack64BitArray (const void *ptr, int size) pushes the `size` elements from the 64 bit integer or double array `ptr` into the buffer.

void CCHUNK::Unpack8BitArray (void *ptr, int size) pops `size` 8 bit integer elements off the buffer and writes them into the array `ptr`.

void CCHUNK::Unpack16BitArray (void *ptr, int size) pops `size` 16 bit integer elements off the buffer and writes them into the array `ptr`.

void CCHUNK::Unpack32BitArray (void *ptr, int size) pops `size` 32 bit integer or float elements off the buffer and writes them into the array `ptr`.

void CCHUNK::Unpack64BitArray (void *ptr, int size) pops `size` 64 bit integer or double elements off the buffer and writes them into the array `ptr`.

The following pack/unpack methods are implemented as inline methods based on the above methods:

void CCHUNK::PackChar (const char c) pushes a single 8 bit integer (char) into the buffer (implemented via `Pack8BitArray(&c,1)`).

void CCHUNK::PackShort (const short s) pushes a single 16 bit integer (short) into the buffer (implemented via `Pack16BitArray(&s,1)`).

void CCHUNK::PackLong (const long l) pushes a single 32 bit integer (long) into the buffer (implemented via `Pack32BitArray(&l,1)`). Can transparently be used with `int` data type because the compiler can apply an implicit cast.

void CCHUNK::PackFloat (const float f) pushes a single 32 bit float (float) into the buffer (implemented via `Pack32BitArray(&f,1)`).

void CCHUNK::PackDouble (const double d) pushes a single 64 bit float (double) into the buffer (implemented via `Pack64BitArray(&f,1)`).

void CCHUNK::PackCharArray (const char *c, int size) pushes the `size` elements of the array `c` of 8 bit integers (chars) into the buffer (implemented via `Pack8BitArray(c,size)`).

void CCHUNK::PackShortArray (const short *s, int size) pushes the `size` elements of the array `s` of 16 bit integers (shorts) into the buffer (implemented via `Pack16BitArray(s,size)`).

void CCHUNK::PackIntArray (const int *i, int size) pushes the `size` elements of the array `i` of 32 bit integers (ints) into the buffer (implemented via `Pack32BitArray(i,size)`).

void CCHUNK::PackLongArray (const long *l, int size) pushes the `size` elements of the array `l` of 32 bit integers (longs) into the buffer (implemented via `Pack32BitArray(l,size)`).

void CCHUNK::PackFloatArray (const float *f, int size) pushes the `size` elements of the array `f` of 32 bit floats (float) into the buffer (implemented via `Pack32BitArray(f,size)`).

void CCHUNK::PackDoubleArray (const double *d, int size) pushes the `size` elements of the array `d` of 64 bit floats (doubles) into the buffer (implemented via `Pack64BitArray(d,size)`).

char CCHUNK::UnpackChar (void) pops a single 8 bit integer (char) off the buffer and returns it (implemented via `Unpack8BitArray()`).

short CCHUNK::UnpackShort (void) pops a single 16 bit integer (short) off the buffer and returns it (implemented via `Unpack16BitArray()`).

long CCHUNK::UnpackLong (void) pops a single 32 bit integer (long) off the buffer and returns it (implemented via `Unpack32BitArray()`). Can transparently be used with `int` data type because the compiler can apply an implicit cast.

float CCHUNK::UnpackFloat (void) pops a single 32 bit float (float) off the buffer and returns it (implemented via `Unpack32BitArray()`).

double CCHUNK::UnpackDouble (void) pops a single 64 bit float (double) off the buffer and returns it (implemented via `Unpack64BitArray()`).

void CCHUNK::UnpackCharArray (char *c, int size) pops `size` 8 bit integer (char) elements off the buffer and writes them to the array `c` (implemented via `Unpack8BitArray(c,size)`).

void CCHUNK::UnpackShortArray (short *s, int size) pops `size` 16 bit integer (short) elements off the buffer and writes them to the array `s` (implemented via `Unpack16BitArray(s,size)`).

void CCHUNK::UnpackIntArray (int *i, int size) pops `size` 32 bit integer (int) elements off the buffer and writes them to the array `i` (implemented via `Unpack32BitArray(i,size)`).

void CCHUNK::UnpackLongArray (long *l, int size) pops `size` 32 bit integer (long) elements off the buffer and writes them to the array `l` (implemented via `Unpack32BitArray(l,size)`).

void CCHUNK::UnpackFloatArray (float *f, int size) pops `size` 32 bit float (float) elements off the buffer and writes them to the array `f` (implemented via `Unpack32BitArray(f,size)`).

void CCHUNK::UnpackDoubleArray (double *d, int size) pops `size` 32 bit float (double) elements off the buffer and writes them to the array `d` (implemented via `Unpack64BitArray(d,size)`).

Versions using unsigned integer types exist too, but are for brevity not listed here.

D.2.2 CNETOBJ

`CNETOBJ` implements a base class that all classes to be transferred using the implemented mechanism must be derived from. It defines macros to *register* the class, which means that its name and a static `Create()` method are stored in a global list. Upon receiving an object this list is then searched for the method to create the object.

It implements a method to send an object to a connected socket and some methods to receive known and unknown objects from a connected socket, but does not handle the connection of the socket by itself. The term *known object* refers to situations where due to an external protocol it is known which object will be arrive next, and the appropriate method throws an error if it actually sees a different object. The term *unknown object* refers to situations where such an assumption cannot be made and any registered object may arrive, and the appropriate methods will then return that object. If an unregistered object arrives all receive methods will return an error. In case of any error the data from the associated object is discarded so that the communication channel is not jammed with junk.

The C++ class `CNETOBJ` is declared in `netobj/netobj.h` and listed in table D.5. It defines abstract `Pack()` and `Unpack()` methods that must be implemented by the user in the derived class. These methods are called with a `CCHUNK` as argument and can use its pack/unpack methods to buffer the data. Upon returning from the `Pack()` method the object is then send over the socket. It also provides a set of methods to deal with the sending and receiving of objects and defines some macros used to register objects.

The following method have to be implemented by derived classes:

virtual const char *CNETOBJ::GetName (void) const = 0x0 returns a unique name for the class (most likely the class name itself). The name is only used for comparisons and may therefore point to static or const memory. The reason for this method despite the general existence of *run time type information* (RTTI) is that the Watcom C/C++ compiler used under QNX did not support RTTI.

virtual void CNETOBJ::Pack (CCHUNK *chunk) = 0x0 is called for serialisation when an object is send. It is passed a `CCHUNK` as argument that has to be used as buffer. The derived class must ensure that all its data is correctly buffered to `chunk`.

virtual void CNETOBJ::Unpack (CCHUNK *chunk) = 0x0 is called for deserialisation when an object is received. It is passed a `CCHUNK` as argument that has to be used as buffer. The derived class must ensure that all its data is correctly read from `chunk`.

The following methods are implemented to support sending and receiving of objects:

void CNETOBJ::Register (const char *name, CNETOBJ * (*createFunc) (void)) registers the class `name` and its `Create()` method in a static global list.

int CNETOBJ::SendSocket (CSTREAMSOCK *sock) calls `Pack()` to serialise an object and then sends the buffer over a socket.

CNETOBJ *CNETOBJ::ReceiveUnknown (void *data, int size) must be called as a static method. It reads a buffer from memory, creates the

```
class CNETOBJ
{
public:

  CNETOBJ (void);
  virtual ~CNETOBJ (void);

  static void Register (const char *name, CNETOBJ * (*createFunc) (void));

  int SendSocket (CSTREAMSOCK *sock);

  // this function 'receives' one unknown object from a memory block.  'unknown'
  // means that the type is not predetermined but the object will instead be
  // created (if the type is registered).

  static CNETOBJ *ReceiveUnknown (void *data, int size);

  // these functions receive one unknown object from a socket.

  static CNETOBJ *ReceiveUnknown (CSTREAMSOCK *sock);

  // the same as above for known objects.  the function must be called from the
  // object to be received.  if the received object is actually a different one
  // a negative error will be returned.  otherwise the return value will be
  // positive.

  int ReceiveKnown (CSTREAMSOCK *sock);

  // these ones will be overlayed by descendants.

  virtual const char *GetName (void) const = 0x0;
  virtual void Pack (CCHUNK *chunk) = 0x0;
  virtual void Unpack (CCHUNK *chunk) = 0x0;
};

#define NETOBJ_DECLARE(TYPE) \
  virtual const char *GetName (void) const {return #TYPE; \ }
  virtual void Pack (CCHUNK *chunk); \
  virtual void Unpack (CCHUNK *chunk); \
  static TYPE *Create (void) {return new TYPE;}

#define NETOBJ_REGISTER(TYPE) \
  TYPE::Register(#TYPE,(CNETOBJ * (*) (void))&TYPE::Create);

#define NETOBJ_DYNCAST(obj,TYPE) \
  (TYPE *) (((void *)(obj)->GetCreate () == TYPE::Create) ?  (obj) :  0x0)
```

Table D.5: CNETOBJ declaration.

corresponding object not expecting a particular object and de-serialises it by calling its `Unpack()` method. If the object is not registered the data is discarded and `NULL` returned.

CNETOBJ *CNETOBJ::ReceiveUnknown (CSTREAMSOCK *sock) must be called as a static method. It reads a buffer from a socket, creates the corresponding object not expecting a particular object and de-serialises it by calling its `Unpack()` method. If the object is not registered the data is discarded and `NULL` returned.

int CNETOBJ::ReceiveKnown (CSTREAMSOCK *sock) must be called as a normal method of an instance of a derived class. It reads a buffer from a socket, uses `GetName()` to ensure that this buffer really contains the expected object, creates it and de-serialises it by calling its `Unpack()` method. If the object is not registered or not of the expected class the data is discarded and -1 returned, otherwise the (positive) size of the object data is returned.

The macros defined are:

#define NETOBJ_DECLARE(TYPE) has to be included in the class definition of a derived class to declare a set of required methods.

#define NETOBJ_REGISTER(TYPE) must be used once at the beginning of each program to register the class in the static global list.

#define NETOBJ_DYNCAST(obj,TYPE) can be used to check if an object is of a certain type or not, similar to dynamic casts in RTTI in the C++ compiler.

D.3 CLABDEV – Control of Laboratory Devices

`CLABDEV` is the communication framework that allows easy and network transparent communication between devices. It is implemented in three C++ classes, `CLABMSG`, `CLABDEV` and `CLABDEVD`.

D.3.1 CLABMSG

Communication between `CLABDEV` and `CLABDEVD` is done by transmitting simple *messages* describing the command and its parameters. These messages are implemented in the `CLABMSG` class, containing native variables for the respective host architecture. The class is derived from `CNETOBJ` and declared in `labdev/labmsg.h` and shown in table D.6. It implements `Pack()` and `Unpack()` methods to encode the message.

```
#define MAXSTRINGDATA 4

class CLABMSG : public CNETOBJ
{
public:
  int _cmd;
  int _intData[8];
  float _floatData[8];
  char *_stringData[MAXSTRINGDATA];
  int _auxDataSize;
  void *_auxData;
};
```

Table D.6: The message class `CLABMSG`: "`_cmd`" holds the command code on sending commands and the return error code on sending replies, "`_intData`" and "`_floatData`" arrays of optional numeric parameters, "`_stringData`" an array of strings and "`_auxDataSize`" and "`_auxData`" a block of auxiliary data that is copied without encoding.

Using the default constructor all values are initialised to 0, resulting in empty strings. If strings are used, they have to be a full copy of the original string (see `strdup()` and/or `malloc()`), not just a copy of the pointer. This is because the same `CLABMSG` instance will be used for receiving the reply, at which point the old contents will be `free()`ed (if non-zero).

The _auxData block of memory can be used to transmit memory whose encoding is otherwise ensured, like JPEG images. If present (non-zero) it must be a block that has been malloc()ed because it may also be free()ed internally.

D.3.2 CLABDEV

The class CLABDEV implements the client side, the part that is requesting services. It is declared in labdev/labdev.h and shown in table D.7. The constructor is *protected* because this class is not supposed to be instantiated directly, but only through a derived class.

```
class CLABDEV
{
protected:
  char *_name;
  CINFOBASE *_ib;
  CLABDEV (const char *name);
  int RequestService (CLABMSG &msg);

public:
  virtual ~CLABDEV (void);
};
```

Table D.7: The client class CLABDEV as declared in labdev/labdev.h. Except for constructors and internal variables it has only one relevant method, RequestService(), that implements the entire message exchange.

Apart from the constructor – which takes the *name* of the service under which the server is looked up in the INFOBASE as argument – CLABDEV has only one relevant method, RequestService(), which is described as follows:

RequestService() has to be called with a CLABMSG message that must have been filled by the user of the derived class. This message is send to the server. When the server completes the command it replies with a message that is returned by the method, deleting (free()ing where applicable) the previous contents (which is why the msg argument is not const).

D.3.3 CLABDEVD

The class CLABDEVD implements the server side, the part that is providing services. The constructor is again *protected* because this class also is not supposed to be instantiated directly, but only through a derived class[4]. CLABDEVD is declared in labdev/labdevd.h and shown in table D.8.

[4] Actually it is an *abstract* class because it also lacks an implementation of Event() that has to be filled in by the derived class.

```
class CLABDEVD
{
protected:
  CINFOBASE _ib;
  CSTREAMSOCK _sock;
  CLABDEVD (const char *name);
  virtual int Allow (unsigned int addr);
  virtual void Event (CLABMSG &msg) = 0;

public:
  virtual ~CLABDEVD (void);
  int Run (const int timeout);
};
```

Table D.8: The server class `CLABDEVD` as declared in `labdev/labdevd.h`. Except for constructors and internal variables it has three relevant methods, that implement the main event loop and a limited access check.

Apart from the constructor – which takes the *name* of the service under which it is registered in the INFOBASE as argument – `CLABDEVD` has three relevant methods, `Run()`, `Allow()` and `Event()` which are described as follows:

Run() must be called in a loop to wait for messages. If the `timeout` argument (in milliseconds) is positive the method will block for at most that time and then return (may be used for keepalive tests). If it is negative the method will block forever.

Allow() is invoked as a *call-back* when a message is received. The `addr` argument is the IP address of the remote host requesting a service. Using this method a minimal safety check can be implemented, only allowing requests from restricted hosts. This method may be overwritten by the derived class when the default behaviour to grant all accesses is not tolerable.

Event() is invoked when a message has passed the above test. This method must be implemented by the derived class. It can use the `msg` argument to implement the service, write the contents of the reply message to `msg` and then return. The reply is then send back to the requesting client automatically.

D.4 CROBOT – Accessing the Mobile Robot System

The entire mobile robot system can be accessed by using a single C++ client class, `CROBOT`, which communicates with a server class `CROBOTD` using the `CLABDEV` framework. Both classes will be documented in this section.

D.4.1 CROBOT

`CROBOT` (see again table 6.8 on page 232) is derived from `CLABDEV` and implements the client interface to the robot's functionality. This interface is all that can be externally seen

from the robot – all the implementation details of the commands are hidden in the scripts and parameters in the INFOBASE. The methods declared in `CMOBILE` are defined as follows (see also the definition of the state automaton on page 219, or tables 7.1 and 7.2 in section 7.1 for comparison):

PickTubeStorage() drives the mobile platform from wherever it is standing to the tube storage for NUNC tubes *without* barcode, moves the arm over the expected position of the marker on the storage, finds the marker using the vision system, compensates the displacement, selects a slot to take a tube from by looking at the state information in the INFOBASE, moves the arm to that slot, uses to vision system to centre on the tube, grasps the tube and retreats the arm into a park position.

PlaceTubeStorage() drives the mobile platform to the tube storage for NUNC tubes *without* barcode and places a tube into a slot, similar to `PickTubeStorage()`, except that a free slot is selected.

PickTubeStorageBarcode() is equivalent to `PickTubeStorage()`, only that the tube storage for tube *with* barcode is used.

PlaceTubeStorageBarcode() is equivalent to `PlaceTubeStorage()`, only that the tube storage for tube *with* barcode is used.

PlaceTubeWaste() drives the mobile platform to the waste bin and drops the NUNC tube into it. This routine does not use any vision and fine-positioning because the waste bin does not have a marker.

HoldTubeSampler() drives the mobile platform to the sampler, moves the arm over the expected position of the marker on the rack inside it, finds the marker using the vision system, compensates the displacement, selects a slot by looking at the state information in the INFOBASE and places a NUNC tube into it. To allow the sampler to work properly, the robot then regrasps the tube below the cap so that it cannot be pulled out of the rack accidentally by the sampler's needle. The arm remains holding the tube upon termination of this method.

TakeTubeSampler() reverts the regrasp sequence from `HoldTubeSampler()` and takes the (now filled) NUNC tube out of the sampler again.

HoldTubeSampler2() is equivalent to `HoldTubeSampler()`, except that the second sampler is used. Since all samplers use the same number of motions to manipulate them only a different set of transforms is needed to describe these motions, hence a different device in the INFOBASE.

TakeTubeSampler2() is equivalent to `TakeTubeSampler()`, except that the second sampler is used.

HoldTubePipette() drives the mobile platform to the pipette, moves the arm over the expected position of the marker attached to it, finds the marker using the vision system, compensates the displacement, optionally shakes the tube a given number of times to resuspend the aliquot and moves the NUNC tube under the needle. The depth how far the needle is inserted into the tube must be given as a parameter to allow for a different treatment of full and empty tubes. The arm remains holding the tube upon termination of this method.

TakeTubePipette() reverts the sequence from `HoldTubePipette()` (except from the vision) and takes the NUNC tube back from under the needle.

OpenCentrifuge() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel (using the same colours as the artificial markers), presses the "open" button to release the lid, grasps under the lid, performs a short force-aborted motion upwards to ensure the lid is really unlocked and opens it using a circular motion. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

PlaceTubeCentrifuge() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel, moves the arm over the cage inside the centrifuge and detects its orientation. Depending on the orientation it temporarily deposits the NUNC tube in a small storage attached to the centrifuge (which again uses visual fine-positioning) and adjusts the cage by pushing onto one of its pads and rotating it with a circular motion so that the free slot is close to the arm, regrasps the tube and inserts it into the slot and retreats the arm. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

PickTubeCentrifuge() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel, moves the arm over the cage inside the centrifuge and detects its orientation. Depending on the orientation it adjusts the cage by pushing onto one of its pads and rotating it with a circular motion so that the used slot is close to the arm and grasps the tube. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

RunCentrifuge() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel and presses the "run" button to start the centrifuge. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

CloseCentrifuge() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel, grasps behind the lid and closes it with a circular motion downwards. The lid is expected to snap locked when falling down. To ensure it is really closed, a short force-aborted motion upwards is performed. If the lid is found to be still open the arm is repositioned over the handle of the lid and slowly pressed downwards. The snapping of the locking mechanism is detected using the force sensor. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

CloseCentrifugeReally() drives the mobile platform to the centrifuge, fine-positions the arm over the pattern formed by the coloured buttons on the front panel, grasps under the closed (?) lid and performs a short force-aborted motion upwards to see if it is really closed. If the lid is found to be still open

the arm is repositioned over the handle of the lid and slowly pressed downwards. The snapping of the locking mechanism is detected using the force sensor. This method is superceded by `LoadAndRunCentrifuge()` and/or `StopAndUnloadCentrifuge()`.

LoadAndRunCentrifuge() optimises one of the most common combinations of the above methods by dropping unnecessary park and unpark motions. The actions performed otherwise correspond to subsequent calls to `PlaceTubeCentrifuge()`, `CloseCentrifuge()` and `RunCentrifuge()`.

StopAndUnloadCentrifuge() optimises another of the most common combinations of the above methods by dropping unnecessary park and unpark motions. The actions performed otherwise correspond to subsequent calls to `OpenCentrifuge()` and `PickTubeCentrifuge()`.

OpenFridge() drives the mobile platform to the fridge, fine-positions the arm over the marker, grasps behind the handle of the closed sliding lid and pushed it sideways to open it. During most of the opening a high force is taken as a jamming lid and the command aborted, but during the final stage the lid is deliberately pressed to the stop.

PlaceTubeFridge() drives the mobile platform to the fridge, fine-positions the arm over the marker, moves the arm over the marker on the rack inside it, fine-positions over the rack, selects a free slot by looking at the state information in the INFOBASE and places the NUNC tube into it. It then retreats the arm. This method is superceded by `PlaceTubeAndCloseFridge()`.

CloseFridge() drives the mobile platform to the fridge, fine-positions the arm over the marker, moves the arm behind the handle of the opened sliding lid and closes it by pushing it sideways similar to `OpenFridge()`. This method is superceded by `PlaceTubeAndCloseFridge()`.

PlaceTubeAndCloseFridge() optimises the most common combination of the above methods by dropping unnecessary park and unpark motions. The actions performed otherwise correspond to subsequent calls to `PlaceTubeFridge()` and `CloseFridge()`.

HoldTubeScanner() drives the mobile platform to the barcode scanner, fine-positions the arm over the marker and holds the NUNC tube above it (which means that the scanner can read the barcode). The arm remains holding the tube upon termination of thus method.

RotateTubeScanner() rotates the tube by a small random angle around its current position over the marker of the barcode scanner. This method is used if the scanner cannot read the barcode, for example if accidentally only the side where the barcode is fixed to the tube with adhesive tape is facing the scanner. The arm remains holding the tube upon termination of thus method.

TakeTubeScanner() takes the NUNC tube from the barcode scanner.

PickCedexStorage() drives the mobile platform to the storage racks for Cedex tubes, fine-positions over the marker, selects a full slot by looking at the state

information in the INFOBASE, fine-positions over that slot and grasps a Cedex tube from it.

HoldCedexPipette() drives the mobile platform to the pipette, finds the marker using the vision system, compensates the displacement, and moves the Cedex tube under the needle. Contrary to `HoldTubePipette()` the depth how far the needle inserts into the Cedex tube is determined automatically by using the force sensor: Once it has detected that the needle has reached the bottom of the tube the motion is stopped. The arm remains holding the tube upon termination of this method.

TakeCedexPipette() reverts the sequence from `HoldCedexPipette()` (except the vision) to takes the Cedex tube from the pipette.

PlaceCedexCedex() moves the mobile platform to the Cedex, fine-positions the arm over the primary marker on the case, moves the arm over the carousel, fine positions over the secondary marker on it and places the Cedex tube into the slot next to it.

PickCedexCedex() moves the mobile platform to the Cedex, fine-positions the arm over the primary marker on the case, moves the arm over the carousel, fine positions over the secondary marker on it and grasps a Cedex tube from the slot next to it.

PlaceCedexWaste() drives the mobile platform to the waste bin and drops the Cedex tube into it, similar to `PlaceTubeWaste()`.

ParkCharger() moves the mobile platform to its charging station and docks to it, waiting for a new sequence to start.

UnparkCharger() undocks the mobile platform from the charging station.

D.4.2 CROBOTD

`CROBOTD` (see again figure 6.5 on page 231) is derived from `CLABDEVD` and implements the server providing the robot's functionality. In addition to the network communication it also contains the complete robot arm control[5], making it quite complex. The declaration shown in D.9 is therefore only a simplified version of the real declaration.

The main components inherited from `CLABDEVD` can still be recognised, along which the basic control flow can be shown as follows:

Allow() is called when a message is received to check if it comes from a legal host.

Event() is then called to process the message. This method basically implements a large *case-switch* as in table D.10.

At this stage the `CLABDEV` framework is left and the implemented robot functionality is called. Starting with `ExecuteScript()`, this functionality is organised hierarchically to map high-level commands (like `PickTubeStorage()`) onto a set of primitives actually accessing

[5]The mobile platform is accessed using the C++ class `CMOBILE`. This class implements a remote control that does not follow the `CLABDEV` scheme. The reason for this is that the `CMOBILE` class was intended to be used in a much wider range of projects which have got nothing to do with the laboratory setup in this project.

```
class CROBOTD : public CLABDEVD
{
private:

  int ExecuteOpen (CRCCLSTACK &stack);
  int ExecuteClose (CRCCLSTACK &stack);
  int Hand (int open);

  int ExecuteCircle (CRCCLSTACK &stack, CARGS &args);
  int Track (POS *pTrack, TRSF *tTrack, POS *pMaster, TRSF *tMasterRel, int flag);

  int ExecuteFabortMove (CRCCLSTACK &stack, CARGS &args);
  int FabortMove (POS *p, double fMax, double diffMax, double diffMaxF) ;

  int ExecuteArmCmd (CRCCLSTACK &stack, CARGS &args);
  int ArmStart (void);
  int ArmStop (void);
  int ArmApproach (CRCCLSTACK &stack, CARGS &args);
  int ArmRetreat (CRCCLSTACK &stack, CARGS &args);

  int ExecuteMove (CRCCLSTACK &stack, CARGS &args);
  int Move (POS *p);
  int SingMove (POS *p);

  int ExecuteMoveJ (CRCCLSTACK &stack, CARGS &args);
  int MoveJ (POS *p);
  int MoveJ (JNTS *j);

  int ExecuteFmove (CRCCLSTACK &stack, CARGS &args);
  int Fmove (FMOVEARGS &fmoveArgs);

  int ExecuteMobileCmd (CRCCLSTACK &stack, CARGS &args);
  int MobileMove (CRCCLSTACK &stack, char *devName);
  int MobileForward (double distance);
  int MobileScaleSpeed (double speedScale);

  bool ExecuteCheckState (CARGS &args);
  void ExecuteAbort (CARGS &args);
  int ExecutePushSpeed (CRCCLSTACK &stack, CARGS &args);
  int ExecutePopSpeed (CRCCLSTACK &stack);
  int ExecuteCenterRegion (CRCCLSTACK &stack, CARGS &args);
  int ExecuteConfirmModel (CRCCLSTACK &stack, CARGS &args);
  int ExecuteAnalyzeModel (CRCCLSTACK &stack, CARGS &args);
  int ExecuteCenterModel (CRCCLSTACK &stack, CARGS &args, char *fName = 0x0);
  int ExecuteAdjustCentrifuge (CRCCLSTACK &stack);
  int ExecuteInstallPID (CRCCLSTACK &stack, CARGS &args);
  int ExecuteRemovePID (CRCCLSTACK &stack, CARGS &args);
  int ExecuteSelectSlot (CRCCLSTACK &stack, CARGS &args);
  int ExecuteLoad (CRCCLSTACK &stack, CARGS &args);
  int ExecuteSetTrsf (CRCCLSTACK &stack, CARGS &args);
  int ExecuteNewTrsf (CRCCLSTACK &stack, CARGS &args);
  int ExecuteAddTrsfTrans (CRCCLSTACK &stack, CARGS &args);
  int ExecuteRandomizeTrsf (CRCCLSTACK &stack, CARGS &args);
  int ExecuteMultTrsf (CRCCLSTACK &stack, CARGS &args);
  int ExecutePrintTrsf (CRCCLSTACK &stack, CARGS &args);
  int ExecuteShake (CRCCLSTACK &stack, CARGS &args);
  int ExecuteWaitForKey (void);

  int ExecuteScript (CARGS &args);
  int ExecuteScriptIf (CRCCLSTACK &stack, CARGS &args);

protected:

  virtual int Allow (unsigned int addr);
  virtual void Event (CLABMSG &msg);

public:

  CROBOTD (flags_t armFlags, flags_t mobileFlags,
           flags_t forceFlags, flags_t visionFlags, char *mobileHost = 0x0);

  virtual  CROBOTD (void);
};
```

Table D.9: A (simplified) declaration of the robot server class `CROBOTD`.

```
void CROBOTD::Event (CLABMSG &msg)
{
  switch (msg._cmd)
  {
  case CMD_PICK_TUBE_STORAGE:
    {
      CARGS args ("sPickTubeStorage");
      msg._cmd = ExecuteScript (args);
    }
    break;

    ⋮

  default:
    msg._cmd = -1; // unknown command
    return;
  }
}
```

Table D.10: The big case switch in CROBOTD.

the mobile platform, the robot arm or the vision system. The topmost layer consists of a number of `ExecuteXXX()` methods, each corresponding to a command in the script language.

ExecuteScript() is the topmost method called by `Event()` whenever a command is received. It corresponds to the script command `call`. It loads a new script and executes it as sub-script. After completition the previous script is continued.

ExecuteScriptIf() corresponds to the script command `callif`. It also executes a sub-script, but bound to some conditions. Currently the translational or rotational part of a transform may be used as condition, so that operations like "*call this is the translation is larger than N millimetre*" are possible.

The following `Execute` methods implement functionalities that do not require calling sub-methods:

ExecuteCheckState() corresponds to the script command `checkstate` and implements the state automaton check. If this method returns anything else but 0 the current script is aborted with an error, which is transferred back to the caller.

ExecuteAbort() corresponds to the script command `abort` and is only used during debugging and when severe errors occur (a device is rotated so much that the robot cannot compensate it).

ExecutePushSpeed() corresponds to the script command `pushspeed` and changes the robot arm speed. The current speed is saved on a stack and the new speed set as the current speed times a factor smaller than 1. Nesting commands is explicitly allowed.

ExecutePopSpeed() corresponds to the script command `popspeed` and restores the previous robot speed by popping it off the stack. Using this way instead of scaling it again with a factor larger than 1 rounding errors can be avoided.

ExecuteCenterRegion() corresponds to the script command `centerregion` and uses the vision system to centre on whatever coloured region is currently closest to the image centre. It does so by modifying a transform that must be present and part of the position equation. The command requires the arm to be in a position so that the camera is approximately centred over the region in question.

ExecuteConfirmModel() corresponds to the script command `confirmmodel` and uses the vision system to assert that the camera is currently positioned over a specific device.

ExecuteAnalyzeModel() corresponds to the script command `analyzemodel` and is used to set a transform to the displacement of the given device, which the camera must be currently positioned over.

ExecuteCenterModel() corresponds to the script command `centermodel` and actually centres over the device by iteratively changing a transform that must be present and part of the current position equation.

ExecuteAdjustCentrifuge() corresponds to the script command `adjustcentrifuge` and is the only command specific for a single device. It compensates the rotation of the centrifuge's cage by applying several "telephone dialing" motions until the desired slot can be reached by the arm. The reason for this operation is that the centrifuge does not stop at a deterministic position, and the reason for a specific function is that the operation is too complex to be implemented by the other commands.

ExecuteInstallPID() does *not* corresponds to a script command, but is used internally by `ExecuteAdjustCentrifuge()` to install a PID controller that is used in subsequent motions to implement force control. This function is an anachronism and should not be used.

ExecuteRemovePID() see above – removes the installed PID controller.

ExecuteSelectSlot() corresponds to the script command `selectslot` and selects an empty/full slot at a storage rack while changing the slot's status, depending on a flag parameter. If no matching slot could be found the command aborts the script and an error is returned to the user. Information about slots is stored in the INFOBASE under the appropriate device.

ExecuteLoad() corresponds to the script command `load` and loads all transforms onto the stack that are specific for the given device. Due to the stack mechanism used these transforms will automatically be deleted if the current script is terminated.

ExecuteSetTrsf() corresponds to the script command `settrsf` and sets a transform to either the value of another transform or explicit values.

ExecuteNewTrsf() corresponds to the script command `newtrsf` and creates a new transform on the stack. As with the transforms loaded by `ExecuteLoad()` this transform will be automatically deleted if the current script is terminated.

ExecuteAddTrsfTrans() corresponds to the script command `addtrsftrans` and is used to add a displacement to the translational part of the given transform.

ExecuteRandomizeTrsf() corresponds to the script command `randomizetrsf` and is used to set the given transform to random translation and orientation values. This command is only useful for debugging purposes.

ExecuteMultTrsf() corresponds to the script command `multtrsf` and is used to multiply two transforms. Can be used in conjunction with `AnalyzeModel()` to first detect a displacement and then compensate it.

ExecutePrintTrsf() corresponds to the script command `printtrsf` and prints the translational part of the given transform. This command is only useful for debugging purposes.

ExecuteShake() corresponds to the script command `shake` and is used to shake a grasped tube the given number of times to resuspend the sedimented cells.

ExecuteWaitForKey() corresponds to the script command `waitforkey` and waits until RETURN is pressed on the keyboard. This command is only useful for debugging purposes.

The following `Execute` methods are based on other methods that do not or not directly correspond to script commands, but are of more general interest so that they have been implemented separately:

ExecuteOpen() corresponds to the script command `open` and uses the `Hand()` command to open the gripper.

ExecuteClose() corresponds to the script command `close` and uses the `Hand()` command to close the gripper.

Hand() does not correspond to a script command. It is the low-level method to open/close the gripper, based on a RCCL primitive.

ExecuteCircle() corresponds to the script command `circle` and moves the arm along a circle specified by the arguments. Internally, it uses the `Track()` method to perform the motion. This command is only used during opening/closing of the centrifuge's lid.

Track() does not correspond to a script command. It creates a virtual arm, moves it according to a position equation and requests the real arm to track that motion according to another position equation. The reason for doing this is that RCCL imposes limitations on the real-time changes that a manipulator is requested to track while otherwise performing a normal motion. These restrictions do not apply if the manipulator is not performing a normal motion, but only tracking a target.

ExecuteFabortMove() corresponds to the script command `fabortmove` and is used to perform a motion with force limits. This method is provided for backward compatibility and should not be used. Instead, the `Fmove()` method should be used.

FabortMove() does not correspond to a script command. It implements the mid-level details for force aborted motions initiated by `ExecuteFabortMove()`.

ExecuteArmCmd() corresponds to the script command `arm`. It is the top-level method for a set of arm related methods. If further parses its command line and calls the appropriate method.

ArmStart() corresponds to the script command `arm start` and is used to switch the arm power on (start the trajectory generator). Explicitly switching on/off the arm power can be used to conserve power and reduce vibrations.

ArmStop() corresponds to the script command `arm start` and is used to switch the arm power off (stop the trajectory generator).

ArmApproach() corresponds to the script command `arm approach` and implements the initial motion sequence from the arm park position to the approach position some distance before the target. This sequence contains – depending on the target – joint interpolated motions with configuration changes to ensure a maximum operational range.

ArmRetreat() corresponds to the script command `arm retreat` and implements the reverse approach sequence back into the arm park position.

ExecuteMove() corresponds to the script command `move` and implements the basic cartesian motion given by a position equation.

Move() does not correspond to a script command. It implements the mid-level details of a cartesian motions to a position equation.

SingMove() does not correspond to a script command. It implements the mid-level details of a singularity-robust cartesian motion to a position equation. Singularity robustness is achieved by plain velocity clipping, which means that the motion is not guaranteed smooth. This method should only be used with a low arm speed.

ExecuteMoveJ() corresponds to the script command `movej` and implements the basic joint interpolated motion.

MoveJ() does not correspond to a script command. It implements the mid-level details of joint interpolated motions. Two versions of `MoveJ()` exist: One for cartesian target given by a position equation and one for targets given as a set of joint angles.

ExecuteFmove() corresponds to the script command `fmove` and implements force affected motions. Both force limits and force compliance can be programmed during a motion according to a position equation. Contrary to `FabortMove()` both types may be active at the same time.

Fmove() does not correspond to a script command. It implements the mid-level details of force affected motions initiated by `ExecuteFmove()`.

ExecuteMobileCmd() corresponds to the script command `mobile` and is the top-level method for a set of mobile platform related methods. If further parses its command line and calls the appropriate method.

MobileMove() corresponds to the script command `mobile move` and implements the mid-level details of motions of the mobile platform. At this level, the platform is always moved absolutely to a device given by its name. The coordinates of this device are looked up in the INFOBASE and the platform is moved using the `CMOBILE` class.

MobileForward() corresponds to the script command `mobile forward` and implements the mid-level details of relative straight forward (or backward, if the distance is negative) motions. It is used only when docking to the charging station because during this motion the collision avoidance of the mobile platform is switched off.

MobileScaleSpeed() corresponds to the script command `mobile setspeed` and is used to set the scale factor of the mobile platform speed.

D.5 CMOBILE

The C++ class `CMOBILE` defined in `mobile.h` and given in table D.11 in a simplified form implements the client access to the mobile platform. It communicates with the control program `mobiled` using a binary protocol instead of the `CLABDEV` framework. Each method in `CMOBILE` corresponds to one or more of the commands in that protocol. The description of these methods is as follows:

The first group of methods deal with low-level motion and localisation issues:

GetPosition() returns the current position.

SetPosition() sets a new position as current position (only used during initialisation).

GetAllMarks() fetches the *current* coordinates (because the coordinates can change slightly) of all (not only the visible) reflector marks and returns their number.

GetAllMarksInitial() fetches the *initial* coordinates of all (not only the visible) reflector marks and returns their number.

GetAllLines() fetches the coordinates of all lines and returns their number.

Stop() stops the current motion by abruptly aborting it (no slow-down phase).

Forward() moves forward a relative distance d (or backwards, if d is negative). During this motion the collision avoidance is switched off because this motion is used to approach the automatic charging station mounted to a pillar.

RotateAngle() rotates the platform on the spot to the absolute target orientation a.

Move() moves the platform to the point (x, y) by first rotating to face towards that point, then driving there on a straight line and finally rotating on the spot to the target orientation a.

MovePoint() moves the platform to the point (x, y) by first rotating to face towards that point, then driving there on a straight line and stopping with whatever orientation it arrives.

IsCompleted() returns a flag indicating whether there currently is a motion or not.

```
class CMOBILE
{
  int GetPosition (met_t &x, met_t &y, deg_t &a);
  int SetPosition (met_t x, met_t y, deg_t a);

  int GetAllMarks (CVEC &marks);
  int GetAllMarksInitial (CVEC &marks);
  int GetAllLines (CVEC &lines);

  // motion related stuff (low-level)

  int Stop (void);
  int Forward (met_t d);
  int RotateAngle (deg_t a);
  int Move (met_t x, met_t y, deg_t a);
  int MovePoint (met_t x, met_t y);
  int IsCompleted (void);
  int WaitForCompleted (void);

  // map related stuff

  class CMAP
  {
    int numLines, numMarks;
    CVEC line, mark;
  };

  int GetMap (CMAP &map);

  // scan related stuff

  class CSCAN
  {
    int numScans, numMarks;
    float sx[361], sy[361];
    float mx[180], my[180], mIdx[180];
  };

  int GetNumScanners (void);
  int GetScannerPosition (met_t &x, met_t &y, deg_t &a, int idx);
  int GetScanPlatform (CSCAN &scan, int idx);
  int GetScanWorld (CSCAN &scan, int idx);

  // motion related stuff (high-level)

  int ComputeError (met_t xe, met_t ye, deg_t ae,
                    met_t *xErr, met_t *yErr, deg_t *aErr);
  int MoveDirect (met_t xe, met_t ye, deg_t ae,
                  met_t &xErr, met_t &yErr, deg_t &aErr);
  int MoveViaIntermediate (met_t xe, met_t ye, deg_t ae,
                           met_t &xErr, met_t &yErr, deg_t &aErr,
                           met_t approachDistance);

  // speed (scale) related stuff

  int GetScale (double &tScale, double &rScale);
  int SetScale (double tScale, double rScale,
                double &tScaleRet, double &rScaleRet);
  int ModifyScale (double tScaleFactor, double rScaleFactor,
                double &tScaleOld, double &rScaleOld);

  // mode related stuff

  MOBILEMODE GetMode (void);
  MOBILEMODE SetMode (MOBILEMODE mode);
};
```

Table D.11: A (simplified) declaration of the `CMOBILE` class used to access/control the mobile platform.

WaitForCompleted() waits until the current motion – if any – has completed.

Currently, only one command to access the map exists:

GetMap() fetches the map with all lines and marks.

To access the laser scanners several commands exist:

GetNumScanners() returns the number of laser scanners (2 in case of the platform used in this work).

GetScannerPosition() returns the position of the laser scanner idx in platform coordinates.

GetScanPlatform() returns a scan from scanner idx in platform coordinates.

GetScanWorld() returns a scan from scanner idx in world coordinates, using the current position to convert coordinates (may be wrong).

Based on the low-level motion and localisation commands are a set of high-level motion commands:

ComputeError() computes the error in platform coordinates between a given target position and the current position in world coordinates. This method can be used to compute an offset for the arm to compensate the positioning error of the mobile platform.

MoveDirect() moves directly to the given target like `Move()`, returning the positioning error in platform coordinates.

MoveViaIntermediate() moves to the given target via an intermediate point a distance `approachDistance` in front of the target as seen by the target orientation like `Move()`, returning the positioning error in platform coordinates. This method is used to minimise the positioning error by avoiding rotations as the last motion component.

GetScale() fetches the current speed scale (1.0 = nominal speed).

SetScale() sets a new speed scale, returning the previous scale.

ModifyScale() modifies the current speed scale by multiplying it with a factor, returning the previous absolute scale. This method is used in combination with `SetScale()` to slow down and later restore the speed of the mobile platform without introducing rounding errors of the speed.

Finally, two more methods can be used to control the basic operating mode (STANDBY or TARGET). In STANDBY mode the motors are disabled and the brakes engaged, so it can be used so save energy and/or ensure that the platform will not move even if it is sent motion commands. The commands supported are:

GetMode() returns the current operating mode.

SetMode() sets a new mode, returning the previous one.

Appendix E

Nonsense

For the very impatient, but sharp-eyed reader a condensed version of the entire thesis is given on the next two pages – enjoy! ;-)