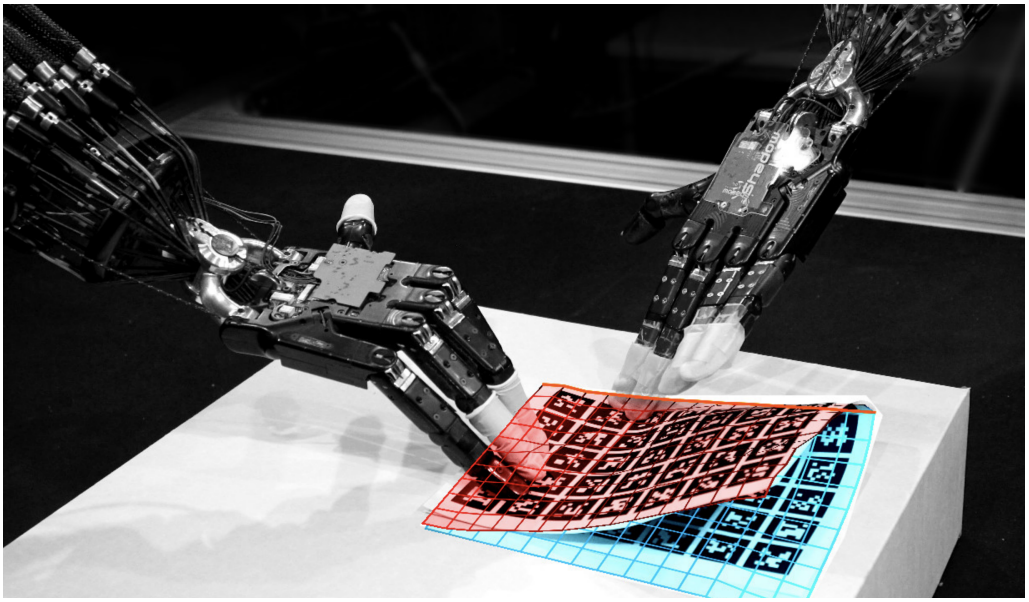
Towards Anthropomorphic Robotic Paper Manipulation



Der Technischen Fakultät der Universität Bielefeld

vorgelegt von

Christof Elbrechter

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

July 2018

Gedruckt auf alterungsbeständigem Papier °° ISO 9706

Für Jonas, Mika und Lotta

Versicherung

Hiermit versichere ich,

- dass mir die geltende Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angefertigt habe, keine Textabschnitte von Dritten oder eigener Prüfungsarbeiten ohne Kennzeichnung übernommen und alle benutzten Hilfsmittel und Quellen in meiner Arbeit angegeben habe,
- dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Vermittlungstätigkeiten oder für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe und
- dass ich keine gleiche, oder in wesentlichen Teilen ähnliche oder eine andere Abhandlung bei einer anderen Hochschule als Dissertation eingereicht habe.

Oerlinghausen, im Juli 2018

Christof Elbrechter

Abstract

The dream of robotics researchers to one day be able to build intelligent multi-purpose household robots that can aid humans in their everyday lives, with the inherent necessity that they are able to interact in general environments, demands that such robots have dramatically improved abilities for real-time perception, dynamic navigation, and closed-loop manipulation. While feed-forward robotic manipulation of rigid objects, ubiquitous in manufacturing plants, is well understood, a particularly interesting challenge for household robots is the ability to manipulate deformable objects such as laundry, packaging, food-items or paper. Given the fact that most objects in our homes are explicitly tuned to be grasped, used and manipulated by human hands, transitioning from traditional robot grippers to anthropomorphic robot hands seems like a necessity. Once we had narrowed our focus to anthropomorphic robot hands, a suitable domain of exploration within the possible set of deformable objects was sought. We chose paper manipulation, which poses many unsolved challenges along the conceptional axes of perception, modeling and robot control. On reflection, it was an excellent choice as it forced us to consider the peculiar nature of this everyday material at a very deep level, taking into consideration properties such as material memory and elasticity. We followed a bottom-up approach, employing an extensible set of primitive and atomic interaction skills (basic action primitives) that could be hierarchically combined to realize ever increasingly sophisticated higher level actions. Along this path, we conceptualized, implemented and thoroughly evaluated three iterations of complex robotic systems for the shifting, picking up and folding of a sheet of paper. Which each iteration it was necessary to significantly increase the abilities of our system. While our developed systems employed an existing bi-manual anthropomorphic robot setup and low level robot control interface, all visual-perception and modeling related tools were implemented from the ground up using our own C++ computer-vision library, ICL. Pushing a piece of paper across a table to a friend is an ability we acquire from a very early age. While seemingly trivial, even this task, which was the first we tackled, throws up interesting hurdles in terms of end-state comfort considerations and the need for closed loop controllers to robustly execute the movement. In our next scenario the paper could no longer be treated as a rigid object, in fact its deformable nature was exploited to facilitate a complex picking-up procedure. Fiducial markers were added to the paper to aid visual tracking and two distinct models were employed and evaluated: a mathematical one and a physics-based one. For our final, fully implemented, system, the robot succeeded in folding a sheet of paper in half using a complex sequence of alternating and in parallel hand movements. Achieving this remarkably difficult feat required us to make further significant improvements to our visual detection setup and a mechanism to model folds in the physics engine had to be implemented. Removing the prerequisite that the paper is covered with fiducial markers was an important hurdle that we overcame using a combination of 3D point cloud and 2D SURF feature registration. Finally, our bottom-up approach to robotic paper manipulation was conceptually extended by the generation of a set of hierarchically organized basic action primitives. The generalization of our approach was verified by applying it to other kinds of deformable, but non-paper, objects.

We believe that a thorough understanding of strategies for dexterous robotic manipulation of paper-like objects and their replication in an anthropomorphic bi-manual robot setup provides a significant step towards a synthesis of the manual intelligence that we see at work when handling non-rigid objects with our own, human hands.

Acknowledgments

The work presented in this thesis was carried out in the Neuroinformatics Group at the Faculty of Technology, Bielefeld University headed by Prof. Dr. Helge J. Ritter and it was supported by the German Service Robotics Initiative (DESIRE) and by the German Cluster of Excellence 277 “Cognitive Interaction Technology (CITEC)”. In the time I developed the software, the systems and the formalisms presented in this thesis, there were numerous people from whom I received an invaluable amount of support, feedback and encouragement as well as constructive criticism and helpful remarks. First of all I would like to thank Helge for enabling me to become a member of his great group and for suggesting the “paper-folding” topic for my thesis. I thank Helge and Dr. Robert Haschke for their supportive, enthusiastic and professional supervision. Knowing that I could call on their combined expert knowledge, ranging from inspiring high level guidance down to constructive discussions regarding low-level implementation details, was a continuous and unwavering aid to me throughout this journey. I would also like to thank the members of the disputation committee, for taking the time to review this work.
I thank all my colleagues from the Neuroinformatics Group and CITEC for their great collaboration across the many projects I participated in. A special and cordial thanks goes to my good friend and longtime colleague, Dr. Jonathan Maycock, who not only re-awoke my interest in the English language but helped me reach a level I would never have expected I could attain. He also was an invaluable and untiring ally when it came to proofreading this 200 page thesis. I thank Dr. Thomas Hermann for snatching me out of his data-mining lecture into the Neuroinformatics Group while I was still a student. The time I worked with Thomas in his ILab allowed me to make my first sustainable steps into the field of computer-vision, which since then has become a passion of mine. I would also especially like to thank Michael Götting, Daniel Dornbusch, Tobias Röhlig and Dr. Abdeldjallil Naceri, with whom I shared my offices with, for all the friendly and helpful discussions and coffee breaks. Furthermore, I would like to thank all the developers who contributed to ICL, with a special shout out to Michael Götting, Andre Ückermann, Andre Justus, Alexander Neumann, Christian Groszewski, Daniel Dornbusch, Dr. Erik Weitnauer, Dr. Felix Reinhardt, Hendrik Hobein, Lukas Twardon, Matthias Esau, Sergius Gaulik, Tobias Röhlig, Viktor Losing and Viktor Richter. In addition, I thank Florian Schmidt for the productive and profitable time we shared while preparing and holding C++ lectures together. An important further thanks goes to Dr. Jens Schmüdderich, Dr. Martin Saerbeck and Benjamin Ludwig for their invaluable impact on my life by sharing lecture halls with me and who made the 8-week full-time learning sessions remain the most enjoyable memories I have of my study time.
Ein ganz besonderer Dank gilt meiner Familie. Ich danke meinen Schwiegereltern, deren kontinuierliche Unterstützung stets einen sehr großen Beitrag für meine beruflichen Freiheiten leistete. Ich danke meinen Eltern und meinen Brüdern für die wohl glücklichste und unbeschwerteste Kindheit, die man sich vorstellen kann und für ihre stetige liebevolle Unterstützung (Mama, Du siehst, es hat sich letztendlich doch ausgezahlt, dass Du mir damals auf dem Anhänger nach der Kartoffelernte immer mal wieder *ganz schwere* Matheaufgaben gestellt hast).
Mein wichtigster Dank jedoch, gilt meiner Frau Saskia, die mit ihrem Vorschlag “Willst Du nicht was studieren?” den eigentlichen Grundstein für meine akademische Laufbahn legte. Sie kümmerte sich nicht nur während, sondern auch nach unserer gemeinsamen Studienzeit stets liebevoll um unsere drei wunderbaren Kinder und hätte sie mir nicht im Alltag immer wieder den Rücken freigehalten, wäre das Entstehen dieser Arbeit nicht möglich gewesen.

Contents

1 Introduction

Robots that can aid humans in their homes to carry out various general tasks have been a long desired goal of the robotics community. However, in contrast to well established and successful production robots used in construction facilities, a large set of additional difficulties has to be solved on the way of porting such robots from their highly controlled industrial environments into household situations. While assembly robots are usually employed in perfectly controlled and coordinated production lanes, households are highly dynamic and non standardized environments that require robots to automatically orient themselves and to autonomously react to unforeseen changes in their surroundings. In addition, production robots are typically extremely specialized in terms of their end-effector-tools and their operation programs. Due to the limited space in our apartments and houses, it is not an option to have multiple units specialized for different tasks. There are some special cases in which smaller robots seem acceptable, such as lawn-mowing robots or vacuum-cleaning robots, however for larger, human sized, robots, it is desirable to only require a single instance that can carry out most different tasks. But what would such a generic robot have to look like? The answer is rather obvious: since such a household robot's main task would be to interact with things that were originally designed and optimized to be used by humans, it will most likely be necessary to endow them with anthropomorphic hardware. Just like in industrial settings, in which each tool-change comes up with significant additional time, space, maintenance and resource costs [Henrich and Wörn, 2012], generic end-effectors are a highly desirable thing to have and the most generic end-effectors that we know of are our own hands. Basically every handle or grip or thing that is supposed to be grasped or moved is dimensioned and tweaked for human hands. Furthermore, considering the number of household tasks that require two hands, a bi-manual robot design seems to be mandatory.

While a large set of possible tasks for an eventual household robot, such as tidying up rooms, emptying the dishwasher or grocery bags, can be formalized by a repetition of the program *grasp object A and put it to place B*, many other tasks require the robot to not only move, but also to manipulate objects. This becomes particularly difficult if the objects that are to be handled have internal degrees of freedom as the robot here not only has to solve its own internal kinematics including joint-limit-aware planning and collision avoidance, but also needs to be able to anticipate the object's kinematics. In addition, many objects that humans commonly deal with can not be described in terms of rigid parts that are connected by joints. Instead, objects such as laundry, cloth, many food items, fluids, plants, packaging, threads or paper have continuously deformable characteristics and therefore need to be handled in a special way.

To achieve this requires a highly complex closed-loop coordination of visuo-haptic perception, planning and motor action schemes employing robotic actuator systems that have typically more than 50 degrees of freedom. This is why most current work is focused on rigid object manipulation, often simplified by assuming rather regular and convex shapes, such as cuboids or spheres, which allow classical planning approaches to be employed.

In contrast, in order to cope with deformable objects whose shapes are likely to change during or even *due to* the manipulation, these approaches must be significantly extended along the three major axes: *perception*, *modeling* and *robot control* (see Section 2.2.1) In order to create a systematic and yet manageable route towards the robotic handling of deformable objects, this thesis is focused on the manipulation of paper, i.e. deformable 2D surfaces with high stiffness and significant ductile properties. The handling of such objects is relevant to several application domains

and poses a rich subset of challenges, such as manipulation under high shape variability, the need for continuous tactile and 3D visual feedback, the involvement of elasticity, *material memory* as a result of bending and folding, and the possibility of complex construction sequences exploiting these properties.
One possible approach for creating highly complex robot control systems is to start with minimal primitive interaction units (basic action primitives) that are then hierarchically combined to create increasingly sophisticated building blocks for interactive robot control and object manipulation. Following this bottom-up idea, we develop iterations of dexterous robot control systems for shifting, picking-up and folding of paper to give an impression of the inherent complexity of creating primitive control systems for seemingly simple paper manipulation actions. Later, this idea is taken up and extended by the conceptualization of a generic system for dexterous anthropomorphic manipulation of paper and paper-like objects.
We believe that a thorough understanding of strategies for dexterous robotic manipulation of paper-like objects and their replication in an anthropomorphic bi-manual robot setup provides a significant step towards a synthesis of the *manual intelligence* [Maycock et al., 2010] that we see at work when handling non-rigid objects with our own, human hands.

1.1 Related Work

The large amount of related disciplines, such as computer-vision and tactile feedback based perception, sensor-fusion, (physical) modeling and simulation, robot planning and feedback-based control, leads to a many possibly interesting papers, articles and theses. Commonly, these contributions have dealt with more restricted types of deformable objects, which can be broadly categorized into i) *1D linear deformable objects*, such as ropes, cables and wires ii) *2D planar deformable objects*, such as cloth, paper or sheet-metal and iii) *3D generic deformable objects*, such as pillows or foamed material.
A typical robotic object manipulation system can be subdivided into the three conceptual components *Perception*, *Modeling* and *Robot Control*. While *Perception* summarizes the acquisition and the processing of all kinds of sensor input, such as cameras, touch sensors and joint-angle encoders, the *Modeling* component defines how the to-be-manipulated object is modeled and how the model's parameters are updated with respect to the processed sensor input. The planning and the actual execution of robot movements is summarized by the *Robot Control* component. In the remainder of this section, general related work for the robotic manipulation of deformable objects is presented and organized with regard to these components. More specific related work is presented in the relevant chapters 3, 4 and 5.

1.1.1 Perception

A robotic system must be able to perceive its world to register where relevant objects are, how its actions affect these objects and to be able to interactively react to external changes. A purely feed-forward system design, where the positions, the orientations and the states of all relevant objects are assumed to be given is not appropriate for the targeted household environment as the robot shares its world with humans, who act as independent and not fully predictable entities. Furthermore, even very small inaccuracies of the actions carried out by the robot accumulate, so that the estimate of the configuration after consecutive manipulation actions becomes inaccurate over time.
The main sense that a system must rely on is vision as it offers a cheap and fast source of information that does not require to touch the manipulated objects and thereby create the risk unin-

tentionally alter its configuration. However, not only visual, but also tactile input is commonly used. The latter is of particular importance in situations in which contact-related occlusions do not allow parts of the objects to be visually detected. Tactile sensor feedback can be obtained by attaching contact pressure sensitive (touch) sensors to the robot's end-effector or by calculating object-to-robot contact forces on the basis of torque-sensors added to the robot's joints. In addition, deformability characteristics such as material stiffness or rigidity can often not be detected without interaction. Instead, physical parameters must be estimated on the basis of the reaction of the object to external forces or to the manipulation itself, which is commonly referenced in literature as *force-based interaction* [Natale, 2003]. In some situations, the estimation of object parameters can require additional *probing interactions* to be conducted prior to the actually intended manipulation.
There are several systems that explicitly acquire and track a geometrically and physically plausible object model, which defines the connection between perception and robot control [Bersch et al., 2011; Elbrechter et al., 2012a; Miller et al., 2011; Schulman et al., 2013a,b].
Even though the availability of such a model greatly facilitates robot planning, most systems employ less specific models that use a topological object representation [Koganti et al., 2013; Twardon and Ritter, 2015b; Wang et al., 2011; Yamakawa et al., 2007] or even make do with the estimation of interest points that can for example be used for grasping [Cusumano-Towner et al., 2011; Maitin-Shepard et al., 2010]. The main reason for this is the inherent difficulty of generic model-based tracking of deformable objects, which is usually achieved by employing fiducial markers [Bersch et al., 2011; Elbrechter et al., 2012a] or specially colored object patches combined with trivially colored backgrounds [Miller et al., 2011; Schulman et al., 2013a,b].
It becomes evident that the complexity of the actual desired interaction is directly linked to both the complexity of the model and in turn also the perception mechanism that is needed. While pinch-grasping towel corners to fold towels can be achieved by estimating grasp points [Maitin-Shepard et al., 2010], dexterous interactions with anthropomorphic robot hands requires more detailed model information. This effect is also reflected by the work that is presented in this thesis. An early system for *shifting paper* (see Section 2.2) was implemented using a very simple rigid paper model that is easier tracked in 2D only. In contrast bi-manual paper folding (see Chapter 5) requires a much more sophisticated model and tracking framework. Specific related work dealing with the detection of paper for robotic paper manipulation is presented in Section 5.1.

1.1.2 Modeling

If we expect robots to move and interact in general environments, a necessary prerequisite is that they are endowed with an internal representation, or model, of the world and objects therein. The model not only formalizes the world mathematically, but also defines model parameters, which reflect physical properties and quantities. In addition, the model describes how these parameters are updated continually as new perceptual input is received. Most robotic systems have hard real-time constraints, necessitating a trade-off between modeling accuracy and computational complexity. Thus, to avoid both conceptual and computational overheads, a system must employ a model that is appropriately tailored to the targeted application. In the case of deformable object models, the intrinsic object dimensionality (1D, 2D or 3D) is often explicitly reflected by the chosen model. That is, when dealing with linear objects, such as ropes, a 1D deformable object model suffices. However, when dealing with paper or cloth a more complex 2D deformable object model is required. 3D models must be employed only for objects that have real intrinsic 3D properties such as sponges or pillows.
Linear deformable models are used in cases in which one of the object's dimensions is much larger than the other two. Most commonly, variants and extensions of classical mass-spring models, internally representing the 1D-object as volumetric tubes of nodes connected by springs are

employed [Jeong and Lee, 2004; Lamiraux and Kavraki, 2001; Pai, 2002]. In contrast, real linear models require additional spring types to be added to account for more realistic bending, stretching and torsion behavior (if needed) [LeDuc et al., 2003; Phillips et al., 2002; Wang et al., 2005] or internally represent the 1D structure as a series of rigid links [Brown et al., 2004; Schulman et al., 2013b; Zheng et al., 2014].
Planar deformable objects, distinguished by a single negligible dimension, can coarsely be split into two classes. Models that allow continuous deformations of the 2D surface in 3D space to be represented, and more abstract models that represent binary deformations such as a 180 degree fold. Due to the specific relevance for the work carried out in the course of this thesis, planar deformable objects are reviewed in more detail later with a focus on bending in Section 4.1.2 and on folding in Section 5.1.2.
For the modeling of general volumetric 3D deformable objects, purely geometrical models such as splines can be used [Terzopoulos et al., 1987]. However, in order to achieve real-time applicability, more recent approaches commonly model such objects physically using mass-spring systems [Duan et al., 2016; Kot, 2014] or, in cases in which accuracy is more important then computational efficiency, finite element methods are used [Paulus et al., 2015].

1.1.3 Robot Control

The third of the three conceptual components is *Robot Control*. This encapsulates all robot-related aspects, such as planning, action sequencing and the actual high and low level control needed to actuate the robot. Robot control to manipulate deformable objects is not only highly relevant for future household robots, but also for industrial and medical applications. Most body parts, such as soft tissue, muscles, internal organs, skin and blood vessels, and also many medical instruments such as elastic needles for suturing, threads, or endoscopes are deformable in nature. In general, once again the dimensionality of the handled objects provides a good structure. Similar to the reduction of computing power necessary to model an object with a lower dimensionality, limitations to the handled objects also allows more specialized and thus more powerful planning algorithms to be created. Without restricting the to-be-handled object type, extended Probabilistic Roadmap Methods (PRMs) are often employed for planning. These generate randomly spread configuration samples with altering deformations and poses [Anshelevich et al., 2000; Moll and Kavraki, 2004, 2006]. In contrast to the planning of rigid object manipulation, the binary collision estimation that is typical for classical PRM-based approaches can be replaced with soft-limits that penalize object collisions according to the depth of the mutual object penetration [Bayazit et al., 2002; Gayle et al., 2005].
When restricting the to-be-handled object to be planar, most applications deal with folding aspects. These include not only folding of cloth/laundry or paper in household situations, but also carton-folding and sheet-metal bending. The planning of folding sequences can be split into three layers:

1. Planning of where to fold the object.
2. Planning of the folding order.
3. Planning of robot movements to achieve this.

Admitting that this suggested structure possibly demands bi-directional interconnections between these layers, the problems are commonly tackled in a separate fashion. Classical mathematical formalisms about origami design [Demaine and O'Rourke, 2007; Lang, 1996] serve well as a basis for point 1 and also partially for point 2. However, for the actual sequencing of folding operations to achieve a desired folding configuration, constraint, collision and manipulability-driven optimizations must be incorporated. The planning of actual robot movements (point 3) very

much depends on the desired goal. Gravity-based folding approaches of cloth [Lakshmanan et al., 2013] often neglect internal material stiffness and thus allow folding operations to be carried out more easily. In contrast, paper folding [Balkcom and Mason, 2008] or in sheet-metal-bending [Gupta et al., 1998] applications, the material stiffness and its plastic behavior has to be explicitly taken into account. Usually, this is achieved by using specialized folding hardware.
It is important to mention that real dexterous robotic paper manipulation using anthropomorphic robotic hands was not properly approached before this thesis' authors work [Elbrechter et al., 2011a] and [Elbrechter et al., 2012a]. An extensive survey on *Model-based Manipulation planning of Deformable objects* concluded the following:

"The dexterous two-handed manipulation required for folding paper as humans do is still an open issue as for a robotic implementation" [Jiménez, 2012].

Additional sources that present systems for the planning of robotic folding sequences are provided along with the actual work chapters for picking-up paper (see Section 4.1.3) and folding paper (see Section 5.1.3).
If the handled object's dimensionality is reduced to 1D, knotting and unknotting tasks [Kudoh et al., 2015; Marzinotto and Stork, 2016; Wakamatsu et al., 2005; Yamakawa et al., 2013] are often investigated. Similar to the handling of planar deformable objects, the restricted object type allows a layered structure to be introduced that decouples the (usually topological) knotting theory from the robot planning steps. A special feature of the knotting/unknotting scenario is the weak plastic behavior of ropes. This leads to the fact that, in theory, unknotting can be achieved by reversely performing knotting steps and vice versa. However, while this assumption is true for topological knot theory, the actual actions that are needed for knotting and unknotting are often completely different. The difference of the actions needed is also reflected by the perceived difficulty. While for example a tightened knot is relatively easy to create, its untying can be a real challenge. In contrast, tying a bow requires a very complex bi-manual interaction but it can be untied by simply pulling one of the rope's loose ends. Robot planning for knotting operations is commonly addressed by including topological consistency checks into the configuration rating of random-motion-based planners [Ladd and Kavraki, 2004; Saha and Isto, 2007].

1.1.4 Two Exemplary Projects

Most of the above related work deals with one or more specific sub-problems addressed in this thesis, but the overall overlap with this work is still rather small. However, there are two projects that were particularly important for this work: Devin J. Balkcom's work on *Robotic Origami Folding* and Pieter Abbeel's Group's work focused on robotic manipulation of deformable objects. Even though Balkcom's idea of a specialized origami folding robot is basically the complete opposite of anthropomorphic robot hands, his work [Balkcom, 2004; Balkcom and Mason, 2008] was very important as its core idea of robotic paper manipulation is very similar to ours. Furthermore, it yielded important insights about the feasibility of robotic paper folding using a specialized robot and his formalisms about the representation of folds and *fold difficultly classes* are very valuable.
Another important project for this thesis was the work done by Pieter Abbeel's Group, which started with *gravity-based folding of cloth*[Van Den Berg et al., 2011]. Also inspired by Balkcom's ideas and by Bell [2010]; Bell and Balkcom [2009], their system is much closer to our work. In contrast to Balkcom, they incorporate visual feedback and they employ a general purpose bi-manual PR-2 robot. However, they moved from paper manipulation to cloth, in particular towels and laundry, which allows them to perform gravity-based folding motions by assuming no material stiffness. After presenting their *Parametrized shape models for clothing* [Miller et al., 2011] and introducing a detection and modeling framework for rope-like objects

[Javdani et al., 2011], they eventually came up with a generalized point-cloud-based tracking framework that, similar to our work, uses a physics-based model [Schulman et al., 2013b]. Their specialization on cloth and rope-like objects has had a strong impact on suturing in medical robotics [Schulman et al., 2013a]. They also provided valuable research towards motion-planning for cloth manipulation and laundry folding [Lakshmanan et al., 2013], learning from demonstration [Lee et al., 2014, 2015a; Schulman et al., 2016] and even force-based manipulation of deformable objects [Lee et al., 2015b].
For the interested reader, there are several extensive reviews and surveys about robotic manipulation of deformable objects [Henrich and Wörn, 2012; Jiménez, 2012; Khalil and Payeur, 2007, 2010].

1.2 Contribution of this Thesis

The intention of this thesis is to provide a thorough understanding of the challenges and the opportunities of dexterous manipulation of paper and other deformable objects using a bi-manual anthropomorphic robot system.
To this end, shifting paper (see Section 2.2), picking-up paper (see Chapter 4) and folding paper (see Chapter 5) were selected, implemented, evaluated and comprehensively discussed. With each more complicated manipulation example, the requirements for *Perception*, *Modeling* and *Robot control* grew, resulting in the development of more and more sophisticated systems for visual paper detection, (deformable) paper models and robotic manipulation frameworks and (feedback-based) controllers.
In addition, the novel Computer-Vision library, ICL, which was developed along with the presented robot systems, is introduced (see Chapter 3). It combines an unmatched combination of Computer-Vision-related functions with general purpose tools for the development of efficient interactive real-time applications including image and point-cloud processing, a fully-fledged GUI-creation and high performance 2D/3D visualization framework as well as in-built physics-simulation support. All applications presented in this thesis were implemented from the ground up in ICL.
We present the first system to ever perform real anthropomorphic picking-up (see Chapter 4) and folding (see Chapter 5) of paper. To achieve this, new detection frameworks were introduced and successively enhanced. Initially based on fiducial-markers, the final detection system is capable of real-time tracking the deformation of a sheet of printed A4 paper as it is iteratively folded in half several times (see Section 6.2). Along with the detection, also the paper model was successively improved. Starting with a most trivial rigid model (see Section 2.2), aspects of bending (see Chapter 4) and even folding (see Chapter 5) were added and the link to update the model on the basis of the perceptual input was continuously refined. The final model (see Section 6.1) can represent and memorize precise folds and creases and suitable methods for automatic fold detection (see Section 6.4) were also developed and evaluated.
With regards to robotic manipulation, the requirements needed to endow an anthropomorphic robot with the ability to shift (see Section 2.2), to pick-up (see Chapter 4) and to fold paper (see Chapter 5) were investigated and analyzed. To this end, we introduced common manipulation primitives and a set of closed-loop feedback-based controllers and we discuss their seamless integration into the existing robot control framework. Finally, a concept of a generic anthropomorphic robotic system based on an extendable set of parametrized *basic action primitives* for the manipulation of paper (see Section 6.5), other kinds of planar deformable objects (see Section 6.5.4) and even 1D and 3D deformable objects (see Section 6.5.6) was developed and discussed.

1.3 Outline

This thesis is organized as follows. Chapter 3 introduces the Computer-Vision library, ICL, which was as a prerequisite for the work carried out here. The following two chapters show the development and the results of the dexterous robotic manipulation systems that were created for picking up paper (see Chapter 4) and for folding paper (see Chapter 5). In Chapter 6 we describe further extensions that were developed in order to show the possibilities and the limitations for a selected set of obvious next steps. Finally, a summary and discussion followed by a future outlook is presented in Chapter 7.
The following paragraphs give a concise overview of the contents of the main work chapters 3 - 6.

Chapter 3: A new Image Processing Library

The work presented in this thesis heavily relies on perception and to this end the Computer-Vision library, ICL, was developed (see Chapter 3). After defining general requirements for the design of software libraries (see Section 3.1), alternative Computer-Vision libraries are presented and systematically compared to each other (see Section 3.2). The comparison even includes benchmarks of common functionalities. Subsequently, design principles (see Section 3.3.1) and a modular overview of ICL's functions (see Section 3.3.2) are given. After positioning ICL in the landscape of the existing libraries (see Section 3.3.4), the components that had a major impact for the work of this thesis are listed and discussed in more depth (see Section 3.4).

Chapter 4: Picking up Paper

Chapter 4 describes the development of a fully-fledged robotics system designed to allow bimanual picking up of a sheet of paper placed on a flat surface. It starts with presenting related work relevant to the picking-up system (see Section 4.1). In addition to work that deals with the modeling of deformable paper and paper-like objects (see Section 4.1.2) and robot systems able to pick-up paper (see Section 4.1.3), fiducial markers and existing detection libraries are focused upon here (see Section 4.1.1). Fiducial markers are included as our paper detection framework (see Section 4.2) is based on a newly introduced marker type (see Section 4.2.1) that facilitates 3D key-point estimation (see Section 4.2.2) In the subsequent modeling section (see Section 4.3), two possible paper models, a purely mathematical one (see Section 4.3.2) and a physics-based paper model (see Section 4.3.3), are presented. The corresponding evaluation section (see Section 4.4) begins with a detailed analysis of the new marker type's detection accuracy (see Section 4.4.1), then the two models are systematically compared qualitatively (see Section 4.4.2) and quantitatively (see Section 4.4.3). After concluding that the physics-based model has more advantages than its mathematical counterpart (see Section 4.4.5), the final system for picking up paper with the robot is presented (see Section 4.5). This is split into a description of the hardware and software setups (see Section 4.5.1), and is followed by a detailed description and discussion of the conducted picking-up experiment. The chapter closes with a structured discussion of visuo-haptic perception (see Section 4.6.1), modeling (see Section 4.6.2) and particular aspects relevant for robot control (see Section 4.6.3).

Chapter 5: Bending and Folding

In this chapter the robotic system is extended to allow a more complex fold to be executed. Section 5.1 provides a thorough literature review of work relevant to paper folding, once again structured

along the three axes: visual detection (see Section 5.1.1), modeling (see Section 5.1.2) and robot control (see Sections 5.1.3 and 5.1.4). The following section introduces a new detection plugin for BCH-code fiducial markers to replace the former custom-designed marker type (see Section 5.2). Section 5.3 describes how the original physics-based model is extended to allow folds to be represented (see Section 5.3.1) and how the model control-law that updates the model with regard to the visual detection must be extended (see Section 5.3.2) accordingly. The following evaluation section (see Section 5.4) first compares the performance of the new BCH-code markers to the custom marker type that was introduced with the picking-up system in Section 4.2.1. Once the superior performance of the new markers is shown, the resulting enhanced paper detection system is extensively evaluated on the basis of a large set of complex paper folding examples carried out by a human (see Section 5.4.2). Subsequently, Section 5.5 describes the development of the robotics system to fold paper. After highlighting the improvements of the software architecture with respect to previous picking up system in general (see Section 5.5.1), new closed-loop feedback-based controllers are introduced (see Section 5.5.3). The final robot control system for bi-manual anthropomorphic paper folding is then described and its results are presented (see Section 5.5.4). The section ends with a discussion (see Section 5.6).

Chapter 6: Advanced Aspects

The last work chapter is dedicated to further extensions that were developed in order to show the possibilities and the limitations of possible next steps. Firstly, an improvement of the paper-model that was developed for the folding experiment is presented (see Section 6.1). While non-axis-aligned folds previously had to be approximated by a regular grid of nodes, the new model can represent arbitrary folds in a precise manner. The following three sections describe the development of a whole new marker-less visual detection system that is based on a Microsoft Kinect Camera. Section 6.2 introduces the new paper tracking system and discusses its strengths and weaknesses by tracking a paper folding sequence. In the following Section (6.3), its major drawbacks are nullified by combining the point-cloud-based tracking approach with a parallel 2D-SURF-feature-based update mechanism. A comprehensive evaluation reveals that this extension elevates the system's tracking accuracy to results that are comparable with the original marker-based tracking system. In Section 6.3.3, the system is employed to track the folding of a sheet of paper with an even more difficult to detect paper texture that shows mostly printed text. A remaining disadvantage of the presented system, the necessity to add fold lines manually, is tackled in Section 6.4. After comparing different metrics for the local fold likelihood along the surface of paper, a particle-based mechanism for detecting *if* a fold was added and *where* it was added is presented, qualitatively evaluated and discussed. Finally, in Section 6.5 a concept of a generic system for dexterous robotic manipulation of paper and other deformable objects, based on a set of basic action primitives (BAPs) is developed. To this end, an initial set of BAPs is created from an exemplary paper aeroplane folding experiment and then iteratively refined and extended by discussing how other common interactions could be realized on the basis of sequences and hierarchical cascades of the existing BAPs. Here, not only paper, but also other planar deformable objects are discussed and a possible generalization to 1D and 3D deformable objects is provided.

2 Rich Research Challenge of Paper Manipulation

The landscape of robotic manipulation is mainly characterized by two orthogonal axes that represent the complexity of the used robot/robot hand and that of the object being handled or manipulated. The term complexity here strongly correlates to the number of DOFs the robot or the object has, but it also encompasses other more abstract ideas of *difficulty*. Figure 2.1 shows a coarse sketch of this landscape and it highlights major blocks of robotics research along the two chosen axes. Of course there are many research projects that address aspects that span several of these blocks and often approaches are generalized by moving up a particular block or deeper to the right. The underlying link to DOFs suggests a weak but significant positive correlation between the two axes. That is, the more complex the handled object is, the higher are the requirements for the employed robot hardware.

The robotic handling of rigid objects is already well understood and thus recent research is largely focused on advanced aspects, such as autonomous planning, collision avoidance, end-stage comfort or learning. However, dexterous handling of rigid objects using anthropomorphic robot hands and closed-loop tactile feedback to allow for example a key to be blindly picked out of a pocket, is still a mostly unsolved topic.

When traversing the landscape further to the right, we approach a research block that deals with objects that in addition to their six external DOFs also have a number, I, of internal DOFs. Such objects can be accurately represented by articulated models that consist of rigid object parts connected by well defined links. Thus the state-space of an articulated object can usually be trivially represented by $\mathbb{R}^{6+I}$. In contrast, the missing rigidity of an object commonly leads to a tremendous increase of the requirements for a perception system.

Interestingly, given the similarity of robots and articulated objects – both are commonly represented by kinematic chains/trees – allows us to bi-directionally exchange formalisms regarding robot planning and the planning of movement sequences for articulated objects. Manipulation of more complex objects and/or using more complex robots is less well-understood.

An even larger challenge arises from the handling of continuously deformable objects. Here, kinematic chains or graphs can only represent the object deformation to a certain level, commonly leading to a trade-off between accuracy and real-time applicability. Therefore, employed models, perception techniques and manipulation frameworks are often customly tailored here to the intrinsic dimension of the manipulated object. Actually, the particular affordances of the handled object classes (1D, 2D or 3D) implicitly links the applications to their intrinsic dimensionality, which makes the exchange of formalisms between these much more demanding. The high complexity of deformable objects necessitates a large set of additional requirements for all parts of the robotic manipulation system.

In particular, the perception components must be extended so that they can handle the continuously changing appearance of the object. Statistical outlier detection methods (such as RANSAC), no longer suffice because of the varying relative transform between object features, and visual feature detection becomes arbitrarily complex as features may be altered, created or even removed during the manipulation. The same is true for tactile perception, as the deformable object is likely to create perception artifacts that arise from the object contact itself. Soft objects not only require highly sensitive tactile sensors, but object specific deformation properties, such as bendability,

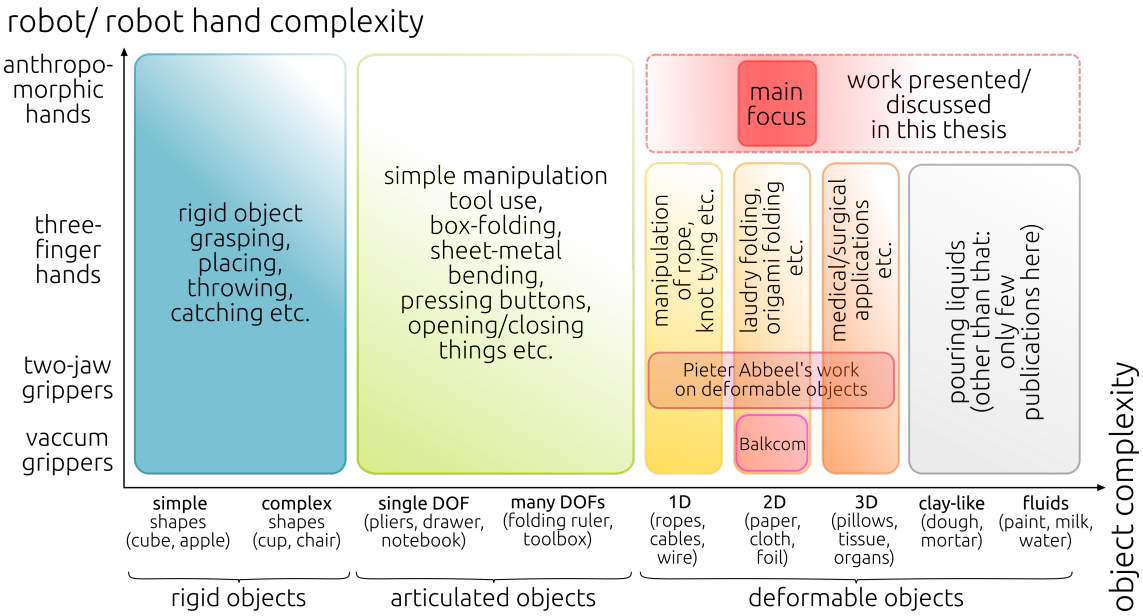


Figure 2.1: Coarse scheme of the landscape of robotic manipulation research. The area is spanned by two complexity axes: the robot and the manipulated object. The background saturation of the sketched areas indicates an idea of the density of research papers for the corresponding combination of robot and hand complexity, which in turn reflects how well understood the particular areas of research are. As landmarks, the areas tackled by the aforementioned projects by Balkcom and Pieter Abbeel are also marked in the landscape.

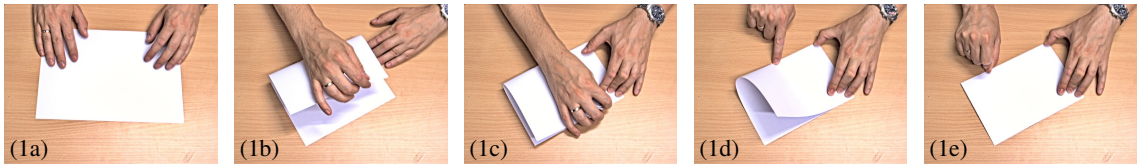
foldability, elasticity, tearability or even plastic behavior require highly sophisticated and physically plausible modeling techniques combined with a robust update mechanism that adapts the state of the model on the basis of visual and haptic perception input. Here, aspects regarding the real-time applicability of a system are very likely to become an important factor. The number of DOFs that are needed to satisfactorily model object deformation drastically reduces the prediction horizon so that systems must counteract the increasing uncertainty by continuously updating and refining their object-beliefs during the interaction.

The bad predictability of the results of actions also tremendously increases the difficulty of possible planning and action sequencing modules. Due to the fact small inaccuracies at the current time can lead to large future errors, action sequencing must be carried out on several abstract levels. Available theories such as knotting (1D) or origami folding (2D) can be employed to arrive at a sequence of desired object movements. On top of that, another layer is needed that sequences robot actions to actually realize the desired movements of object regions. Internally, even rather primitive actions, such as *establish contact at a given object position*, are likely to require their own built-in real-time closed-loop visuo-haptic feedback channel. The task gets conceptionally even harder when dealing with clay-like objects that allow material parts to be taken off.

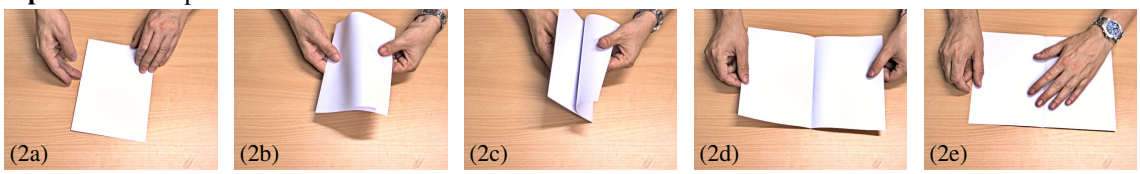
While in the literature, there is already a solid basis of research papers dealing with robotic manipulation of deformable objects, there is only a very small amount of publications that employ anthropomorphic robot hands for this purpose.

This thesis aims to provide concepts and solutions to tackle this thus far only barely touched area in our research landscape, and in particular focuses on deformable object manipulation using anthropomorphic robot hardware. Within this field, we chose the manipulation of paper, i.e. 2D objects that combine continuous deformability with plastic behavior. In the context of future household

Step 1: Fold the paper in half along the short edge and harden the crease.



Step 2: Undo the previous fold.



Step 3: Diagonally fold one corner towards the center fold and harden the crease.

Figure 2.2: First instruction steps to build a simple A4 paper aeroplane.

robots, the ability to be able to manipulate paper and paper-like objects will be very important if we want them to help us with many everyday tasks. Through origami theory, there exits a large theoretical background of human paper folding which is likely to become applicable to robotics once a set of primitive interaction building blocks such as *fold along that line* or *turn the paper up* is available.

One of the first interesting interactions a child learns with paper is to build an aeroplane. Figure 2.2 shows the first few steps of this task and it herewith reveals a surprisingly rich set of highly coordinated hand movements that require a vast range of manipulation skills that extend from the low level hand-eye coordination using sensitive tactile feedback to rather high-level action planning and sequencing. A possible first step towards replicating parts of these manipulation skills on an anthropomorphic robot could be to identify and extract re-occuring interaction patterns, which could serve as a basis for the definition and implementation of *basic action primitives* (BAPs). Such primitives could be generalized, re-used and cascaded to realize more complex interaction units and even whole manipulation sequences. In order to arrive at the first step (1a), a unit is needed to position the paper appropriately on the table top. To this end the ability to shift and rotate the paper in-plane is required. The next two steps (1b,c) assume that we are able to grasp a corner of the paper, to fixate another part, to relatively move the grasped part and to align corners or edges of the paper. In addition, the hands must switch fixation points during the interaction and perform several highly dexterous movements while coordinating several fingers simultaneously. Closed-loop visual feedback is not only needed to perform the actions in a feed-forward manner, but also to continuously ensure that the hand actions actually lead to the desired object movements, which is of particular importance for the alignment of the paper edges. Furthermore, many actions, such as grasping and fixating parts of the paper, explicitly require tactile feedback to adjust contact forces that either avoid slippage or, as for creating the crease in (1d,e) explicitly allow slippage. While the next steps, (2a-e) introduce a set of new interactions such as displacing the aligned layers of the paper to separate them (2b) or the two handed straightening (2d), they also show, that a set of already introduced units (such as fixating) can be re-used.

This description, even in condensed form, illustrates the richness and complexity of the chosen domain of dexterous robotic paper manipulation.

Before embarking on our journey to endow anthropomorphic robot hands with paper manipulation abilities it was necessary to first list a set of requirements, written here written as a list of questions,

that drove the conducted research. They emphasize the enormity of the undertaken challenge and indeed illuminate the scope of the work carried out in this thesis.

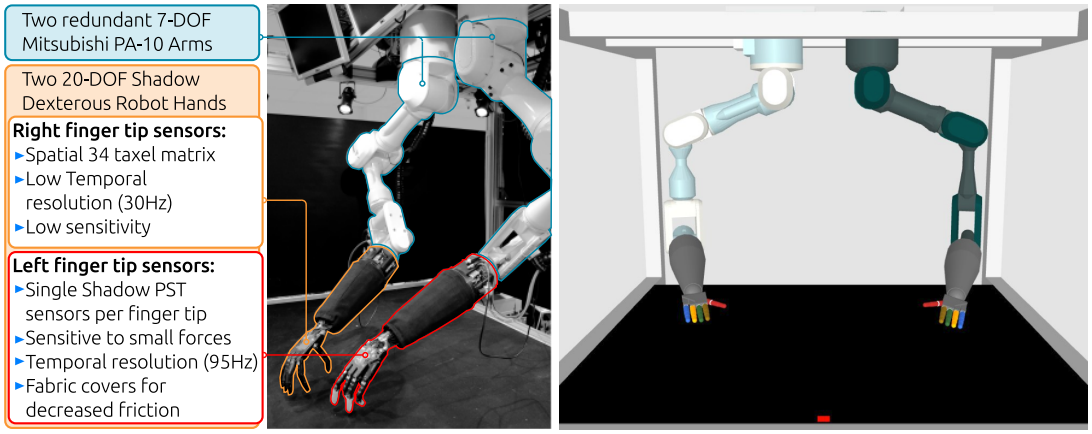
- What software framework can serve as an overarching basis to optimally combine the different domains of computer-vision, tactile perception, (physical) modeling and robot control/planning?
- Can existing tracking mechanisms be extended to provide reliable visual features on deformable 2D objects or do we need whole new methods?
- What requirements has a model and what physical properties must be explicitly handled to achieve a sufficient tracking performance?
- How can we update the model state with respect to perceptual input?
- How can we handle the large amount of occlusions that must be expected (occlusions from the robot hand/ folded parts of the paper that occlude other parts)?
- How explicitly must folds be modeled, and what does that mean for the modeling and perception module?
- How can we model shape memory and plastic behaviour?
- How can we include tactile feedback in the context of action primitives and under given real-time constraints?
- What difficulties arise from using anthropomorphic robot hands and what advantages are offered by their dexterity?
- Can we develop a minimal set of basic action primitives, such as *fixate at position*, *grasp at position* or *move to position*, which can serve as a basis to create complex action sequences?
- Can basic actions be (hierarchically) combined to create a versatile interaction toolkit for robotic paper manipulation?
- What extensions are needed to generalize such a toolkit to manipulate other kinds of deformable 2D objects?
- Can we generalize this further to handle 1D/3D deformable or even simpler rigid objects?

The first of these questions is addressed in the following by presenting our chosen hard and software setup (see Section 2.1). After that, a system for paper shifting is presented, which could be used to prepare a to-be-manipulated paper to arrive at a situation such as in Figure 2.2(1a).

2.1 Hard and Software Prerequisites

For all the work carried out for this thesis, we restricted ourselves to a fixed hardware setup consisting of two robot arms and hands. On the software side, the existing in-house developed robot control framework was also utilized. However, whenever the existing interfaces did not allow a certain behavior to be realized, the required functionality was added to the framework in a generic and re-usable fashion[1]. The computer-vision-related solutions that were created are based on the in-house developed C++ computer-vision library ICL[2]. Similar to the employed robot control

[1] The Author thanks Dr. Robert Haschke for his valuable support in this regard.
[2] This thesis' author implemented more than 80% of its source code.



(a) Robot setup (b) Robot server simulation

Figure 2.3: The Bielefeld Curious Robot Setup **(a)** The robot setup consists of two Mitsubishi PA-10 robot arms, each equipped with a pneumatically controlled Shadow Dexterous Robot Hand. (b) Screenshot of the robot server simulation tool. The simulation is used for collision avoidance and for debugging.

framework, re-usable functionalities that were developed for specific applications were extracted and thus became native constituents of ICL.

2.1.1 The Bielefeld Curious Robot Setup

All robotic systems that were implemented for this thesis are based on the robotics hard and software framework of the *Bielefeld Curious Robot* setup [Lütkebohle et al., 2009] (see Figure 2.3a). The hardware setup consists of two 7-DOF Mitsubishi PA-10 robot arms that are mounted from the top downwards. Together with an antagonistic pair of 20-DOF Shadow Dexterous Robot Hands, the whole setup is designed to resemble a human's arm-kinematics including movable shoulder joints. The arms are attached to the upper center of an aluminum frame. A wooden plate, clad with black cloth is used as worktop.
The software framework is a robot-server architecture that offers command-line and network interfaces for intuitive robot control and trajectory planning in both task and joint space. The server internally extrapolates the robot motion in simulation (see Figure 2.3b) in order to avoid collisions with the setup's security cage and the worktop plate as well as robot self collisions to avoid both self-damage and injuring nearby people. The server also allows joint-angles to be read out and it features some basic transform calculations, such as querying the transform of a given robot limb or defining a logical tool transformation. An important drawback of the system is the fact that both arms and both hands are controlled separately, i.e. it does not provide kinematic solving for a kinematic tree that reaches from the mount of the shoulder joint to the fingertips. Instead, the arms are commonly controlled in task-space mode by one instance of the server, while in parallel, a second *hand-server* instance is used to control the finger-joints. This is a severe disadvantage, in particular, when performing bi-manual actions, which is why recent work has been focused on extensions that allow a more general kinematic model to be used. During the work for this thesis, however, this feature was not available. Therefore, the used bi-manual setup employs two logical[3] hand and arm severs, that are connected via XCF-middleware [Fritsch and Wrede, 2007] using one or several *Active Memory* [Bauckhage et al., 2008] instances (AMIs). The underlying

[3] Internally a single server application can handle two hands or two arms at once, logically separated.

concept of AMI support is, as the name suggests, a memory layer that can be used to store, update and remove global application data at run-time, similar to classical black-board approaches. However, in the context of the robot control system, the AMIs are basically used as event-routers that allow participating processes to send and to subscribe to XML-based events. A more general use-case of AMIs is given in Lütkebohle et al. [2009]. A very practical feature is the fact that event subscriptions can be endowed with an XPath-expression that allows incoming events to be filtered not only based on their type, but also on their content. However, this feature also comes up with a fundamental disadvantage for the AMI-approach. Since there is no native support of event *scopes*, such as provided by the Robotics Service Bus middle-ware [Wienke and Wrede, 2011], all occurring events are delivered (via spread-multicast), unpacked (string to XML) and matched (against all XPath subscriptions) in all participating clients (PCs and processes), which leads to a high network and processor usage. To overcome resulting CPU-load issues, event scopes must be emulated by using several dedicated AMIs for specialized purposes.
For state-based robot control, the software-framework provides a dynamic and XML-based hierarchical state machine (HSM) implementation [Ritter et al., 2007] that is deeply integrated with the robot-servers as well as with the XML-based event-driven robot server communication and inter-process data exchange. The XML-based HSM description syntax not only allows a state/-transition hierarchy to be defined, but it can also be supplemented with built-in imperative robot control statements that allow very self-contained HSMs to be defined. In situations where the built-in code does not provide enough flexibility, Python-code can be natively embedded. A later version, as it was used for the robot folding experiments (see Chapter 5), additionally allows HSM description files to be nested, i.e. *sub-HSMs* that implement certain robot behaviors can be re-used by employing simple *include*-statements. By allowing text-based variable substitutions during the inclusion of an external HSM, this mechanism becomes as powerful as calling parametrized functions.
During the course of this thesis, several requirements that arose from the development of paper-manipulation systems were directly implemented as extensions of this software setup.

2.1.2 The Image Component Library (ICL) for Visual Perception

All visual-perception and modeling tasks that were implemented for the systems presented in this thesis were done using the C++ computer-vision library ICL (Image Component Library). ICL's development started in 2006 in the Neuroinformatics Group of Bielefeld University as a replacement of a formerly used in-house library called *Basic Vision Layer*. During the development of the applications for robotic paper manipulation, ICL was continuously improved and extended along several axes. A core idea of ICL is, that it allows applications to be implemented *in* ICL so that most of the time no extra software libraries are needed. We will show that ICL, mainly developed and implemented by the author of this thesis, is powerful enough for the development of highly complex interactive computer-vision applications, including point-cloud-processing, 3D-rendering and even physics simulations. ICL is open-source and it has been used in many projects within and outside of our institute. ICL mainly provides features that are in general also provided by other, more commonly known libraries, such as *OpenCV* for computer-vision, the *Point-Cloud-Library* (PCL) for point cloud processing, *Eigen* for Mathematics, the *Visualization Toolkit* (VTK) for real-time 3D visualization, the *Qt-Frameworks* for GUI-creation or the *Boost-Libraries* for general purpose programming tools. However, instead of requiring a programmer to get to know many of these libraries, each having its own special types, naming conventions or even mandatory-to-use implementation paradigms, ICL provides most of the commonly used functionalities of all of these libraries and does so without the need to *re-invent the wheel*. Instead, ICL wraps around or adapts to existing libraries when necessary in order to present the programmer with a uniform and consistent interface that allows applications to be implemented with a minimal amount of

boiler-plate code. However, it still maintains the speed and the flexibility of the wrapped libraries and it provides seamless native access to the underlying wrapped software back-ends if needed.
The ICL-based demonstrator systems that are presented in this theses do not only justify ICL's existence, but also strongly underline its widespread capabilities, its performance and its unique versatility for the development of complex and interactive real-time computer-vision applications. ICL was continuously improved and extended to match the requirements that arose from the systems developed for this thesis. Due to its fundamental impact on these systems, Chapter 3 is dedicated to it and its background, design principles, placement within the landscape of other computer-vision libraries as well as a comprehensive listing of its features and benchmarking results are presented. In particular, features that were crucial to this thesis are highlighted.

2.2 Shifting Paper as an Entry Point

In order to sensitize for the inherent complexity of robotic paper manipulation, an initial early experiment, conducted in 2009, will operate as a basis for a discussion about the requirements, the predictable difficulties and the possible limitations of such systems.
As a first entry point for the robotic manipulation of paper-like objects, *shifting paper* was chosen as one of the most simple interactions possible. While the actual interaction logic for shifting paper was developed and implemented in a straight-forward and naive manner, familiarization with different scientific fields, such as computer-vision, robotics, sensor-fusion and software-engineering as well as the examination of available software frameworks and toolkits took center stage during the development. To slightly increase the difficulty of the system, it was decided to not only translate the paper to a target position, but to also align the paper's in-plane rotation with a desired orientation. It is important to mention that the chosen shifting scenario explicitly allows the paper to be modeled as a rigid object. That paper is inherently deformable comes into stark focus in the rest of this thesis.
In order to endow the robot with this rather simple ability, a non-negligible set of challenges has to be solved. Firstly, the system must be enabled to detect the position and orientation of the paper. Once the system *knows* where the paper is and how it is rotated, the robot hand must establish contact with the paper. To this end, an appropriate hand and forearm pose as well as a method to perform the contact with the correct force must be found. It is likely that a too weak contact force leads to slippage resulting in the paper being not shifted correctly. Conversely, pressing the paper too strongly against the table top could damage the robot hand. However, manually controlled tests that were initially conducted, showed that the compliant nature of the used Shadow Robot Hand [Shadow Robot Company, a] yields an inherent hardware-based compensation mechanism for a good range of contact forces. For the rotation of the paper, an extra difficulty is to distribute the contact force equally enough to be able to use two or more touching finger to increase the leverage effect during the rotation movement.
Another required system component is the planning of the robot movement. While the constraint-solving for the control of the robot's joint angles is accomplished by the robot control interface, the high-level movement trajectories must be planned, computed and serialized explicitly. An important factor for this are the kinematic limitations of the robot that do not allow us to perform every rotation task in a single step. In cases in which this is not possible, the paper rotation must be split into several smaller rotation movements, each consisting of a contact-less *reaching-back* movement, followed by an actual forward-rotation movement while maintaining contact. Due to the fact that the actual achieved rotation and translation is likely to differ from the intended one, the paper pose must be tracked during the movement, which gives the system a pose-based visual servoing [Hutchinson et al., 1996] character.

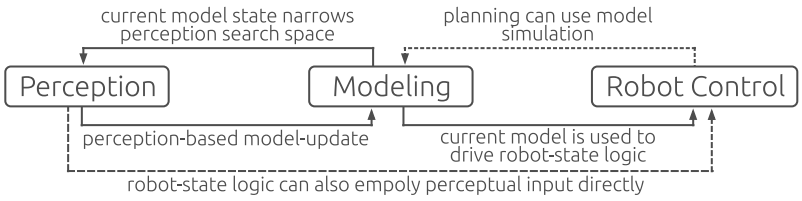


Figure 2.4: Generic three-fold scheme of a robot manipulation system. The *Modeling* component serves as a basis for information exchange between *Perception* and *Robot control*.

2.2.1 Perception, Modeling and Robot Control

As initially motivated, robot manipulation systems can be generally subdivided into the three conceptual main components *Perception*, *Modeling* and *Robot Control* (see Figure 2.4). The perception unit includes the acquisition and the processing of sensor input received from cameras, touch sensors, joint-angle encoders and other possible sensors. The input data is then used to update the state of a scene or object model. While in this thesis, the *scene* is usually represented by a single static support plane only, the development of a model of (foldable) paper that can be updated by visual input is one of its major contributions. The term *model* is employed as a two-fold expression, which is used for both, a *model class* describing *how* the paper is parametrized and a particular *model instance* defined by its underlying class and a current parameter set. Whether in the course of this thesis one or the other is referenced, is either said explicitly or it is made clear from the context. The *Modeling* component includes the used model class as well as one or several instances of it and it describes how the model is to be updated on the basis of the perceived input. The *Robot Control* component defines how the robot is actuated depending on the current perceptual sensor input and the current model state. This includes a task-oriented state-logic, planning and the actual actuation of the robot in joint, task or object space. The low-level joint-limit-aware robot control that solves inverse kinematics for task or object space control, resolves singularities and avoids robot self-collisions using an internal representation of the robot's geometry is given by the used robot server framework (see Section 2.1.1)
Figure 2.4 shows how the three components are interconnected. The information gained from the Perception component is passed to the Modeling component in order to update the current model. In turn, a feedback-link propagates the model state back to the Perception component facilitating a narrowing of the local feature search-space in the next processing step. In addition, the current model is also used as input for the Robot Control component. Here, not only can the robot-state logic compute robot-movements depending on the current model state, but also a possible planning unit could use the model for simulation if needed. As not all sensor data is necessarily fused into the model, the Robot Control component has direct access to the sensor data. This link could e.g. be used for the integration of image-based visual servoing controllers [Hutchinson et al., 1996].
The structuring of a robotic manipulation system into these three fundamental building blocks not only works for the rather simple paper shifting system presented here, but also intuitively scales towards the much more complex applications that are presented later in this thesis (See Chapters 4, 5 and 6).

As for the paper-shifting system, a major part of the implementation of the system has become obsolete by now, because for most of the components, more general and yet more accurate and faster methods have become available in ICL or in the robot control framework. Therefore the actual implementations of the sub-systems are considered to be implementation details and are thus only coarsely presented. However, it is important to underline that this initial rather simple approach sufficed to successfully endow an anthropomorphic robotic system to shift paper on a tabletop.

Perception

The perception building-block of the paper shifting system internally consists of two parts: the visual RGB camera-based paper detection module and the processing of the tactile finger-tip data on the robot hand, which is used to determine whether proper contact with the paper has been established. The latter was trivially implemented by a simple global maximum fingertip pressure threshold that implicitly indicated an appropriate contact had been established between the robot hand and the support-plane.
For the visual detection of the paper, an edge and corner-based 2D image-space tracking system was implemented that provides fast tracking rates at an acceptable robustness in the presence of occlusions. By assuming that the sheet of paper is the only bright blue object in the scene, a trivial RGB-distance based segmentation, followed by morphological filtering allows a paper-edge image to be efficiently computed. The edge and corner features were tracked locally in the spatio-temporal domain, i.e. edges and corners are searched for in the vicinity in which they were located in the previous processing frame. At startup and in cases in which the local search is detected to have failed, a global re-initialization module is activated. In order to map image pixel positions and paper orientation in the image space to the robot's task space, camera calibration is needed. To circumvent the necessity for proper camera calibration, which was, at that time, not natively available in ICL, a hand-tuned homography H that maps image pixel coordinates to 2D task space points $(x, y, 0)$ on the support plane ($z = 0$) was used.

Modeling

In this 2D-scenario, a rigid paper model that is represented by the paper-edges is sufficient. The actual edge positions are formalized by a static 2D-paper dimension and a 3D-paper pose (x, y, θ), describing position and orientation of the paper. Due to the omitted real camera calibration, the actual task space position as well as the orientation of the paper in task-space are computed using the homography H. In order to define a target pose for the shifting of the paper, a mouse-based GUI application was developed that allows a virtual target paper to be moved and rotated in a drag & drop manner.

Robot Control

Even for the simple task of shifting paper, a variety of issues had to be handled. Given, the current paper pose (x_c, y_c, θ_c) and the target paper pose (x_t, y_t, θ_t), the translational shifting and the rotational shifting had to be separated. While the translation of the paper was assumed to be possible using a single shifting motion, the rotation of the paper required an extra logical layer be introduced. First, the system had to be enabled to decide whether to optimally turn the paper in clock-wise or in counter-clock-wise direction, which was solved in the naive straight forward way of selecting the direction that results in a smaller rotation delta. However, it turned out that due to the mounting of the robot arm and the low hand elevation angle that is needed to distribute the finger contact force appropriately, the robot could only kinematically perform rather small paper rotation movements of $\Delta\theta \approx 30°$. Experiments revealed that due to the hand's rigid palm construction, a proper finger force distribution can only be achieved by aligning the palm with the worktop. Two main reasons are responsible for this. First, the – in comparison to a human hand – low dexterity of the thumb and second, the kinematically unfortunate placing of the thumb's touch sensors when the hand is open. To avoid damaging the hand by using too much contact force, the touch-sensors of all fingers including the thumb have to be monitored, which in turn makes it mandatory to align with the worktop. The palm elevation angle is the direct result of

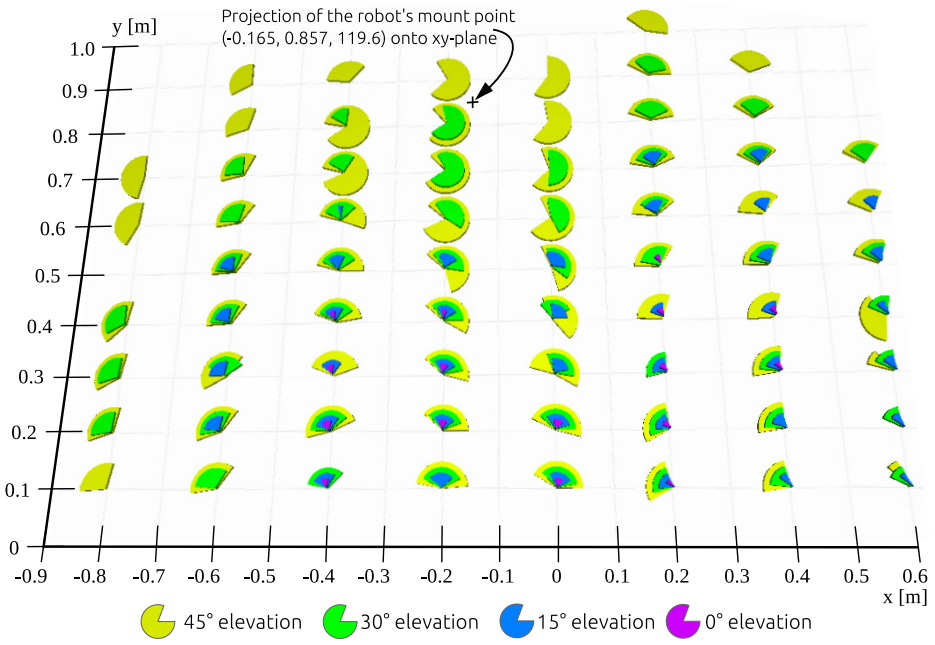
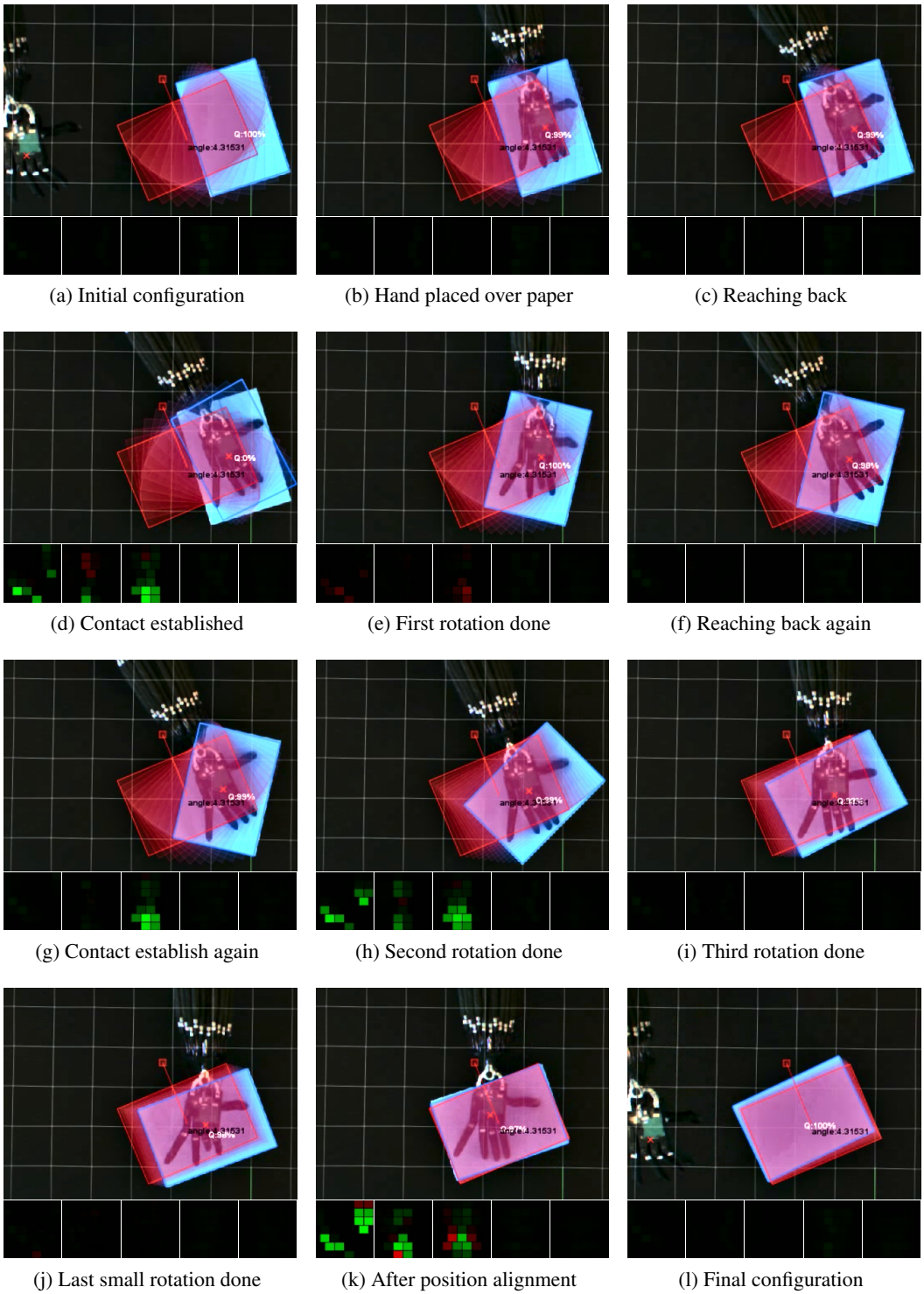


Figure 2.5: Results of a reachability study that shows a schematic of the robot setup's table top plane ($z = 0$). Each of the sampled positions (pie centers) was attempted to be reached by the right index finger tip using different forearm elevation angles (pie colors) and different approaching angles (orientation in the xy-plane). For each position and elevation angle, the robot was moved to that position with orientation 0 (parallel to the negative y-axis). From here, the robot was then rotated around the finger tip along the world z-axis in both clock-wise and counter-clock-wise direction as far as kinematically possible. The maximally accomplishable rotations are indicated by the visible parts of the pie discs.

the forearm elevation (angle between the forearm and the xy-plane) and the vertical wrist flexion. In order to optimize the combination of these two angles, an extensive reachability study was conducted using a simulated arm server (see Figure 2.5). The most obvious result is the fact, that the arm maneuverability directly scales with the elevation angle. Quantitatively, it reveals that the rotation capabilities with zero elevation (purple pie-pieces) are extremely weak even in the sweet spot around $(-0.2\ m, 0.2\ m)$ and literally non-existent at many other positions. The wrist joint limits of the Shadow hand are specified with $[-45°, 30°]$ [Shadow Robot Company, a], but due to a persistent defect in the used hand's wrist muscles or tendons, the actual maximum negative upwards wrist flexion that was maintainable turned out to be in the order of $-15°$ only. Therefore for the final shifting prototype a forearm elevation of $15°$ was used, so the resulting reachability was similar to the one indicated by the blue pie-pieces in Figure 2.5.
After aligning the orientation of the paper in a set of iterative steps, the final translational shifting could be implemented in a straight forward manner.

2.2.2 Results

The results of the shifting experiment are presented in Figure 2.6. In the initial state (see Figure 2.6a), the target paper pose was already defined so that a relative rotation around about $40°$ and a translational shift of about 10cm is required. After approaching the paper (see Figure 2.6b), the first rotation sequence consisting of *reaching back* (see Figure 2.6c), *contact establishment* (see Figure 2.6d) and *forward rotation* (see Figure 2.6e) is performed. This is iterated until the target rotation is reached (see Figure 2.6f-j). Once the orientation is aligned, the paper is not directly released. Instead, it is shifted towards the desired target position (see Figure 2.6k) to fulfill the



(a) Initial configuration (b) Hand placed over paper (c) Reaching back

(d) Contact established (e) First rotation done (f) Reaching back again

(g) Contact establish again (h) Second rotation done (i) Third rotation done

(j) Last small rotation done (k) After position alignment (l) Final configuration

Figure 2.6: Results of the robot shifting experiment. For of the selected sub-steps, a top-view of the robot setup (top image) is shown. As an overlay, augmentations are added that show the current robot end-effector position (red cross), the desired paper pose (red rectangle) and the current paper detection result (blue rectangle) including the pose estimation certainty Q (white text). Below each of the images, the local responses of the five finger tip touch sensors are visualized (from left to right, thumb to little finger). Green pixels correlate to increasing or stable force values, as the force magnitude is mapped to the intensity of the color green. In contrast, pixels that correspond to tactile pixels with a temporally decreasing contact force value are tinted red.

desired paper pose in both orientation and position. At the end (see Figure 2.6l), the paper is released and the robot hand is moved to its resting position.
The visual perception of the blue paper works well throughout the experiment. Only very rarely occlusions by the hands lead to a loss in tracking of the paper, which, however, never causes any severe issues as the system accurately recognized the fact that the paper is lost by returning a zero certainty value (see Figure 2.6d). The closed-loop nature of the system, introduced by re-estimating the paper pose after each rotation/shifting movement, would allow a possible extension of the system to temporarily decrease the amount of occlusions by partly withdrawing the robot hand in such cases. The hand-tuned pixel-based rigid model reflects the real observation perfectly well.
The robot control mechanism allows the paper pose to be aligned with an acceptable accuracy.

2.2.3 Discussion

A major difficulty of the system is the contact establishment procedure, which is implemented as an explicit perceive-action loop in the robot state HSM. In each iteration step, the hand is moved down by 1 cm until a certain maximum touch-sensor value is registered. This has several disadvantages for the achievable operation speed[4], which is caused by the internal signal latency and by the enforced limitations put on the robot acceleration and deacceleration. The low sensitivity of the right robot hand's touch sensors led to a poorly predictable force read-out, which had to be compensated by moving the hand slowly and by waiting some tenths of a second after each movement. The unfortunate placing of the thumb-tip's touch sensor[5] made this even more difficult, requiring us to decrease the force sensor threshold. This in turn increased the likelihood of exiting the contact establishment loop too early, so that the resulting friction did not allow the paper to be shifted at all. Even with the chosen low threshold the thumb broke once during the experiment. To avoid these issues we later decided to extend the robot server framework by a set of closed-loop feedback controllers (see Section 5.5.3). These allowed faster feedback cycles to be achieved by avoiding explicit perceive/act loops or by porting a part of the feedback cycle natively into the server architecture. In addition, one of the new controllers employs the displacement of joint angles to a reference posture to estimate the contact force.
A second difficulty arose when the paper was finally released. While after the final shift (see Figure 2.6k) both the orientation and the position alignment was very accurate, releasing the paper produced an unwanted movement which led to, in particular, an error in its final position (see Figure 2.6l). This effect was caused by the poorly controllable inherent flexibility of the pneumatic-muscle-controlled Shadow robot hand. Even though all pneumatic muscles were emptied before performing the releasing upwards motion with the hand, the remaining tension between paper and hand results in an unintended movement of the paper during this step. As the robot hand's compliant nature is responsible for this issue, it cannot be avoided that easily in software. A possible solution would be to learn an optimal trajectory for the releasing motion, based on active exploration. However, while the high hardware failure probability and maintenance costs make such an approach expensive, it seems not feasible to achieve an appropriate modeling accuracy to solve this task on the basis of a physical simulation. A combination of both approaches that employs a well-tuned physics engine to bootstrap the exploration performed with the real robot would possibly provide a good trade-off here. Alternatively, the use of motor-controlled robot hands, such as the Shadow Motor hands[6] [Shadow Robot Company, b], would make a software-based compen-

[4] The whole shifting sequence took about two minutes.
[5] The limited flexibility of the inner hand makes the thumb touch the ground with the side of its tip if the palm is not parallel with the surface touched.
[6] These were recently integrated into our robot setup to replace the formerly used Shadow Dexterous Hands but they were not used for the work done in this thesis.

sation of this effect much easier.
Even though the planning was shown to provide good results for the presented shifting scenario, there are possible improvements that could be made. The conducted reachability study (see Figure 2.5), which provided valuable insights about possible rotation angles, was only taken into account qualitatively. That is, the results were used to figure out that an elevation angle of 15° would allow the single rotation steps to be at least 30° if carried out around the center of the workspace. A more sophisticated planning module would incorporate the resulting reachability manifold to optimally exploit the robot arm's mobility. Given an energy metric, such as the anticipated time or trajectory length of an action, a proper minimization could result in shifting the paper to a most comfortable position for rotating, followed by a final positioning of the paper. In this context, the *end-state-comfort* effect [Short and Cauraugh, 1997] could be considered. Humans often perform actions in such a way that the final postures of the actions are *comfortable*. The effect can be linked to the bio-mechanical properties of the human locomotor system that is more accurate, more energy-efficient and can produce higher forces and velocities in the mid-range between its joint-limits (See also *middle-is-faster-effect* in [Rosenbaum et al., 1996]). While servo motor driven robots usually perform equally well across their whole joint-range, an action planning module that tries to maximize the robots mobility at the end-state of an action would allow proceeding actions to be more quickly started.
An even more complex shifting approach would require a kinematics solver that can deal with the whole kinematics tree from the robot's shoulder mount to its finger-tips[7]. By these means, in contrast to the implemented decoupled control of arm and hand, smooth and much more natural hand and arm movements could be produced.

[7] This feature is not natively available in the used robot control framework (see Section 2.1.1).

3 A New Image Processing Library

Professional craftsmen know that a toolbox filled with a well chosen set of high quality tools is a major prerequisite if they are to finish handcraft jobs quickly and professionally. The tool-box metaphor fits astonishingly well for software libraries. In addition to the most direct interpretation that a software library contains a set of functions and classes that are used as *tools* by the programmer, the metaphor can be extended to aid the understanding and the definition of requirements for software libraries.
The most useful toolbox is not necessarily the one that contains all possible tools. Indeed certain tools can be found much more easily and quickly if only those tools that are really necessary reside in the toolbox. Furthermore, it speeds up the search process for a specific tool considerably, if related tools are grouped together into smaller sub-units. For software libraries, this means that adding another function or class or adding another parameter to a function does not necessarily increase the library's quality. If a functionality is, however, mandatory for the library, the developer should try to group it appropriately with other related functions.
In general, and borrowing again from the tool-box metaphor, a good software library should endow the programmer with the right tools for fast and professional software development.
In order to facilitate the research carried out for this thesis, a new software framework was developed. The Image Component Library (ICL) is a versatile computer-vision framework that allows for the easy creation of complex and interactive computer-vision applications written in C++. ICL not only includes a large and hierarchically structured set of computer-vision-related functions and classes, but also a well chosen collection of tools that facilitate application development. The most important design principle used in the development of ICL was that it should allow programmers to implement applications *in ICL*. This means that in particular beginner programmers can concentrate on learning C++ in combination with ICL, rather than having to get insights into several other libraries, each providing separate aspects of their applications. Furthermore this helps to reduce the programming overhead arising from the data type conversion necessary to exchange data between different library-interfaces.
This chapter is organized as follows: first requirements for computer-vision libraries are given and explained (Section 3.1). Subsequently, alternative computer-vision libraries, particularly the well known OpenCV library[1], are presented, discussed and coarsely compared with respect to the requirements that were introduced (Section 3.2). The full set of design principles that took center stage during the conception process of ICL as well as the library's main features are then presented in Section 3.3. The chapter ends with a more detailed introduction to the tools that were most useful for the work carried out in this thesis (see Section 3.4).

3.1 Requirements

A generic definition of the requirements for software libraries in general is provided by Gibbs et al. [1990] and Atkinson et al. [1997], who discuss, among other topics, class layout, correct usage of inheritance and variable visibility. Although correctness, defined here as a function doing exactly what it specifies it does, is a very important aspect of a software library, detailed investigations

[1] http://opencv.org

into the correctness of all libraries presented in this chapter is outside the scope of this thesis. It is assumed that all libraries are correct. Rather than breaking down all library components in detail, we try to combine the large set of requirements in order to obtain three, partly contradictory, sub goals: *ease of use*, *speed* and *function volume*. In order to explain this, once again the tool-box metaphor can be employed: tool-boxes are used to perform tasks more easily, professionally and quickly without the need for other tools not contained in the tool-box.

3.1.1 Ease of Use

Along with speed (see Section 3.1.2), *ease of use* is an extremely important requirement for computer-vision libraries. This not only includes an intuitive API. Instead, it starts with the availability of the software, and how easy it is to actually become able to use the library. Therefore, *ease of use* starts with a website that provides necessary information to download, build and install the software library. Furthermore less obvious features, such as platform independence, price or the availability of the software under the terms of an open-source license can contribute significantly to its ease of use. Libraries that are only available for special platforms might be difficult to use or to integrate into existing setups. The requirements of a desired software license type can even make it impossible to use libraries that are only available under the terms of more restrictive license types – in particular proprietary closed-source libraries. Some libraries are even available for different programming languages while providing similar function signatures. A common combination is to wrap parts of an efficiently programmed C++ library into Python. This can speed up the software development cycle significantly due to the omitted compilation cycles.
Coming back to the more obvious features that facilitate the usage of a software library, an intuitive and user-friendly API including documentation is essential. The quality is influenced by many different aspects such as a consistent naming convention, the complexity of function and class interfaces and the correct use of inheritance and other advanced programming techniques. Yet it seems impossible to define a metric that measures the quality in general. However, the readability of the produced code is usually a good indicator of the quality of an API, because source-code is usually written once but read many times during the development process.

Library vs. Framework

A *software library* in general provides a set of re-usable types, functions and classes. A *software framework* usually wraps around a set of corresponding software libraries, but also provides tools that facilitate the creation of applications based on these libraries. These additional support utilities allow for connecting, controlling and maintaining software components on different levels. On the implementation level, additional support classes can be provided to manage an application's control flow. On the platform level, special compiler and linker systems can help users obtain an executable application. In an optimal case, software libraries should be usable even without using all other tools provided by the overarching software framework. In particular if software components are only usable with an associated control flow strategy, they might be difficult to integrate with other frameworks that have similar restrictions to other control strategies.
The availability of a software framework can significantly reduce the amount of overhead on both the implementation side and the compilation side of applications, in particular for beginner programmers.
Even though its name suggests that ICL is a library, the definition above leads to the conclusion that ICL is in fact a framework. However, as ICL's name was already well established in its community, it was decided to not adapt it. Therefore, in this thesis it will be referenced the *ICL framework* whenever its framework aspect is to be emphasized.

Self Containment

Learning how to implement computer-vision applications typically consists of two parts, learning how to program in C/C++ and familiarizing oneself with computer-vision. For beginner programmers, familiarizing themselves with a new software library or framework can be a difficult and time consuming task. At some stage, it is always helpful to implement one's own applications in order to learn how units are supposed to be connected and how changing parameters affect the direct output of a unit and the output of its successors. This part can be significantly more difficult if other external libraries must be learned in parallel. While using test image files as input might suffice at first place, the possibility to implement interactive applications with real-time camera image input and real-time visualization can significantly speed up the learning rate.
For an optimal learning curve, a library should be as self-contained as possible, allowing programmers to write applications *in* that library, rather than implementing them in a programming language *using* that library. Taking this into account, external libraries should only be needed for very special tasks, resulting in not only data-type conversions between library interfaces being avoided, but also the associated computational overhead. By mainly using the naming conventions of a single library, the readability of the produced source-code is also enhanced.

3.1.2 Speed

In many computer-vision applications, processing speed is one of the major benchmarking criteria, where often the constant factor in the complexity analysis of an algorithm distinguishes whether a system has real-time-capabilities or not. In particular, in real-time applications that perform tasks such as object-tracking, fast processing rates can also significantly improve the processing results, because short temporal intervals between successive frames lead to less frame-to-frame motion. The speed requirement is usually the reason why computer-vision-libraries are written in machine-close programming languages such as C or C++. Sometimes there is a conflict when deciding to write efficient code that is not very user friendly or more intuitive code that can result in a trade-off for the designer of the library.
In particular basic commonly used operations such as image-acquisition, linear filters, data type and color conversions or connected-component-based segmentation should be tuned to consume as little resources as possible. Among manually written efficient source-code, there are several general ways to speed up code. When not using C or C++, wrapping natively compiled C/C++ functions can already lead to a significant speedup. In the C/C++ domain, algorithms can often be accelerated using SIMD-instructions. This is, however, not always possible, not completely platform independent and usually difficult to implement. Also multi-threading (e.g., using the OpenMP-framework) can be used to speed up tools on multi-core machines. However, not each algorithm can easily be implemented in a multi-threaded manner, and often caching issues lead to a decrease of the processing speed rather than the desired increase. A newer option for speeding up algorithms is called GPGPU[2], where algorithms are implemented in a highly multi-threaded manner. However, this leads to even more complex code and additional time latencies arising from the necessity to transfer data between the system's RAM and the graphics card's memory.
The speed requirement covers two aspects: First, the library functions should be implemented as efficiently as possible while optimally hiding the complexity of the underlying implementation. Second, the library should provide interfaces to easily extract data in such a way that custom algorithms can be optimized manually if necessary.

[2] General-Purpose computing on Graphics Processing Units

3.1.3 Function Volume

In addition to *speed* and *ease of use*, a library is of course supposed to contain a large set of task related functions. Function volume refers not only to the number of functions in a library, but also to the scope of the functions in a library. To speed up the finding of a specific function, a library should optimally only have functions that are required. Which functions are actually needed obviously depends on the targeted application, however, a hierarchical or indexed library structure that makes it easy to find commonly needed functions can significantly reduce the overhead arising from searching the library for specific functionality. In the worst case a programmer would re-implement a function that already exists because they could not find it in the library. An intelligent use of function parametrization and templating, object oriented programming and inheritance can contribute significantly to a slimmer library interface.

Function Levels

In the following the distinction between low-, intermediate- and high-level functions will be drawn as is commonly done in the literature [Sobrevilla and Montseny, 2003]. Even though there is no generic or clear definition of these levels, they are used to express on which level corresponding sets of functions are to be found in a vision application's scheme.

Low-level functions are assumed to work directly on the raw image pixel data. Examples are commonly used basic operations, such as copying or converting image pixel data, and image filters, such as linear filters and morphological operators. Usually the output of a low-level function is an image.

Intermediate-level functions will in general work on images as well. In contrast to low-level functions, here the output is usually an intermediate representation of the image content, such as regions, key-points or features.

High-Level functions make use of low- and intermediate-level functions to extract some high-level information from an image. Examples are face-detectors and object or person trackers.

Common Functions

Dependent on the targeted sub-domain of computer-vision, the function set of a particular library can have a justified imbalance towards very specific tools. While some libraries are completely focused on providing classical 2D image processing tools, other libraries might mainly focus on 3D vision and point cloud processing. Most of the libraries presented in the next section focus on only one or two particular function levels (see Section 3.1.3). In the comparison section (see Section 3.2.7) , for each level a set of commonly used functions is defined.

3.2 Alternative Computer Vision Libraries

Before ICL is presented, some common alternative computer-vision libraries are discussed. It is worth mentioning that the development of ICL started in 2006 and the availability and quality of

alternative computer-vision libraries has changed significantly since that time. The most comparable computer-vision library is definitely the well known OpenCV[3] library, which has become a quasi-standard in the research community with almost 50 thousand active users. Not completely independent from OpenCV are the Intel®Integrated Performance Primitives (Intel®IPP)[4]. The Intel®IPP libraries are proprietary libraries with a huge set of highly optimized functions for different domains, including computer vision. OpenCV can optionally be linked against Intel®IPP in order to speed up a set of functions internally. Another commonly used, but proprietary, computer-vision library is Halcon[5]. Also proprietary, but usually not used in real-time applications is the image processing toolbox in MathWorks' Matlab. An interpreted and easy-to-learn syntax, plus a large set of functions makes it a good choice for image processing beginners. Examples of other less well known, but free and open-source computer-vision libraries are the CImg library[6], the VxL library[7], the RAVL library [8] and the CCV library[9]. Over the past few years 3D cameras, such as the Microsoft Kinect, have become affordable, making 3D point cloud processing a common domain in computer-vision research. Therefore, although not completely comparable to the other libraries, the Point Cloud Library (PCL) [Rusu and Cousins, 2011] will also be looked at.

3.2.1 OpenCV

OpenCV is the most well known library for computer-vision. It has been developed with a focus on real-time computer-vision applications. Its origins can be traced back to 1999, when it was originally developed under the name *Image Processing Library* (IPL) at Intel®by Gary Bradski [Bradski and Kaehler, 2008]. Later, the development was split into OpenCV and IPL. The latter is now part of the Intel IPP libraries (see Section 3.2.2). OpenCV's IPL origin is readily seen in the name of its main image data type *IplImage*. Almost a decade later in 2008, OpenCV's original developers came back together at Willow Garage, where its development was accelerated leading to a new major version (2.0) in 2009.
The library contains over 500 optimized computer-vision algorithms, plus an even larger set of support functions. It contains low-level algorithms such as linear image filtering and linear algebra, but also ready-to-use high level tools for camera calibration or face detection. OpenCV provides both standard and state-of-the-art algorithms mostly from the domains of computer-vision and machine learning.
OpenCV is split into several libraries, each containing specific tools for image-processing (imgproc), user-interfaces (highgui), camera calibration (calib3d), 2D feature detection (features2d), object detection (objdetect), machine learning (ml) and GPGPU-based computer-vision (gpu and ocl).
Originally OpenCV was implemented in C. Later, parts were implemented in C++ while it still provided a C-interface. Since version 2.0, a new C++ interface is available that also provides a new main image data type implemented in an object-oriented fashion. Even though the C++ interface makes it much easier to write shorter and more self-explanatory code, its has not only brought advantages. Even now, not all parts of the C-based library are available in C++, which means the developer sometimes has to go back to the C interface. This is of course possible, due to the fact that C++ is almost a complete super-set of C, but it leads to confusing code. Furthermore, porting existing systems from the C interface to the C++ interface can also lead to an extensive implemen-

[3] http://opencv.org
[4] http://software.intel.com/en-us/intel-ipp
[5] http://www.mvtec.com/halcon
[6] http://cimg.sourceforge.net
[7] http://vxl.sourceforge.net
[8] http://ravl.sourceforge.net
[9] https://github.com/liuliu/ccv

tation overhead. In addition, OpenCV has wrapper libraries for Java and Python. In June 2015, the new major version 3.0 was officially released. Here, along with some general internal adaptions in the whole library structure, many new functions were introduced, and the OpenCL-based acceleration was significantly extended.
OpenCV is available for all main platforms[10] and even the mobile Android platform. It is well documented and many free online tutorials and examples for most different types of applications exist. Furthermore there are a number of good books, such as [Bradski and Kaehler, 2008] devoted to it.

3.2.2 Intel IPP

Intel®IPP is a proprietary set of highly optimized libraries for signal processing, image processing, small matrix algebra and cryptography. It is available for the main platforms, and it is free for private use[11]. It provides a C-interface which leads, due to the lack of function overloading in C, to a set or many thousands of systematically named functions. All functions that are provided are highly optimized with different types of SIMD instruction sets, such as MMX, SSE and AVX. A dynamic run-time linking process automatically detects the instruction sets supported by the current machine and selects the fastest supported implementation. In this way older and non-Intel®platforms are supported.
Intel®IPP is well documented and includes code snippets that can usually be tested in a cut-and-paste manner. Furthermore, paying customers can make use of professional online support. A well structured hierarchy of functions eases the learning curve for beginners. Except for a few support types such as point, size and rectangle structures, it usually works on standard data pointers directly, which enables programmers to use most Intel®IPP functions without expensive data-type conversions.

3.2.3 Halcon

Halcon is a proprietary computer-vision library, provided by MVTec Software GmbH. It provides bindings for C, C++, C#, Visual Basic and Delphi and it is available for the main platforms. It is packaged with the integrated development environment HDevelop, which provides its own script language. Programs, prototyped in HDevelop can access existing native code modules and also be exported to one of the supported programming languages. Halcon is optimized for professional users and use cases such as production surveillance. It is advertised to contain more than 1800 operators for different sub domains, such as blob analysis, morphology, matching, measuring, identification, calibration, bar-code reading and OCR-tracing [12]. Internally, the library is optimized with both SIMD- and GPGPU-based implementations.

3.2.4 Matlab Image Processing Toolbox

Matlab is a well known and very powerful tool for all different kinds of data processing. It provides its own simple-to-learn interpreted programming language yielding both functional and imperative programming features. While the code interpretation is comparably slow, most built-in functions are highly optimized. A native function interface that allows for implementing function back-ends in C makes it possible to make applications real-time-capable. Matlab can be extended with more

[10] Windows, Linux and Mac OSX
[11] The license terms explicitly mention that research is not private use
[12] http://www.mvtec.com/halcon

than 30 different toolboxes for many different domains of data analysis, such as signal processing, statistics and different sub-domains of image processing and computer-vision. Matlab also provides functionality to create GUI-based interactive applications. Most toolboxes also provide module-definitions for the graphical programming environment Simulink. The current version also contains GPGPU-optimized implementations. Due to the simplicity and readability of Matlab's script language it often replaces pseudo-code in research papers and scientific books. Matlab is available for the main platforms.

3.2.5 Less Common Libraries

A collection of less common vision libraries that are still in use and under development are now presented.

CImg

The CImg Library is a free computer-vision library developed by David Tschumperlé under the terms of the CeCILL or the CeCILL-C license, which is comparable to the well known (L)GPL license. It is mainly written in C, however its central image structure is a C++ class. Even though it was mainly developed by a single author, it contains a large set of functions that range from standard low-level image operations such as linear filters to complex 3D rendering tools. It also directly contains methods for image visualization and methods to create interactive GUI-applications. The core CImg library is shipped in a single header file, where all functions are implemented in an inline fashion. Unfortunately the header file makes extensive use of very long preprocessor macros. Only functions from a few very special domains are provided as modules, again in shape of header files. Optionally, CImg can be augmented by set of 3rd-party libraries. In order to provide a simple-to-use image library optimally suited for programming beginners, one of its design principles was to explicitly avoid the use of extremely generic template-based interfaces. Another design principle was to enable programmers to write complex programs with only a few lines of code, which, however, lead to the fact that many functions have cryptic acronym names.
The development of CImg started in 1999 as a part of the main author's PhD thesis and it is still under development. Since 2003, CImg is hosted at sourceforge and it is compatible with all the main OS platforms.

CCV

The CCV image library is the newest example discussed in this section. It has been developed in C by Liu Liu since 2010 and it is available as a GitHub project and it is written in C. In contrast to the other libraries, CCV does not concentrate on providing low-level image processing tools. Instead it contains a set of classical and state-of-the art intermediate- and high-level tools such as image-feature generators, face-detectors and 2D object and person trackers. Its main idea is an overarching cache architecture that stores intermediate result images in order to avoid a duplicated execution of pre-processing operations. Its public interface is packed into a single C-header file that provides the signatures of all types, functions and definitions. Given the – in comparison to the other libraries – much smaller amount of provided functions, this seems comprehensible. By providing a set of examples and unit tests, beginners can quickly develop their own applications. CCV is available under the terms of the BSD 3-clause license. The documentation does not explicitly mention the supported platforms, but it was successfully tested under Linux.

RAVL

The development of the Recognition and Vision Library RAVL started at the University of Surrey, UK. RAVL is written in C++ and it is provided under the terms of the LGPL license. It can be run on Windows and Linux. Its advertised main features are thread-safety, powerful I/O, and Java-Like easy-to-use class interfaces that focus on enabling programmers to write readable source code. It also comes with its own makefile system allowing for the maintenance of both small and large scale projects.
RAVL provides a vast set of low-level utility classes to facilitate rapid application development. Furthermore, it contains a large set of low and intermediate level image processing functions as well as a framework for image visualization and interactive user interfaces.

VxL

The Vision-*something*-Libraries VxL aims to be a light and fast computer-vision library. It is written in C++ and contains functions for many computer-vision disciplines, such as numerical algorithms, 3D-vision, structure-from-motion, classification and feature tracking. VxL is hierarchically subdivided into smaller libraries. Along with a built-in testing framework, the core-library, which itself is split into two levels, is the basis for a large set of contributions from other institutes. VxL provides its own, not publicly available license and it supports Windows and Linux. An easy installation on Ubuntu-Linux via *apt-get install* is possible.

3.2.6 Point Cloud Library (PCL)

In addition to traditional 2D cameras, cameras that provide 3D images – point clouds – have become affordable over the last few years. While only a limited number of research institutes could previously afford an expensive SwissRanger camera provided by the Mesa Imaging company[13], the availability of Microsoft's Kinect camera in 2010 introduced an affordable device for capturing real-time 3D point cloud data. Since then, the Point Cloud Library (PCL) has become the standard library for 3D point cloud processing in the research domain.
PCL has been maintained by Willow Garage since 2010. It contains a large set of low- and intermediate level functions and utility classes, which are distributed over several modules, such as filters, I/O, registration and segmentation. PCL is completely developed in C++ and provides GPGPU-based optimizations for some components. It is developed very actively by a large community including many known institutes and it supports all main platforms.

3.2.7 Comparison

Before the implemented image processing framework ICL is introduced, the existing alternatives, along with ICL, are compared with respect to the requirements formulated in Section 3.1. Table 3.1 mirrors the three fold structure of Section 3.1 by providing comparisons with respect to *ease of use*, *functionality* and *speed*[14]. The most distinguishing features of the presented libraries are perhaps given by their license types. While Intel ®IPP, Matlab and Halcon are proprietary libraries including guaranteed professional support, the others[15] have open source licenses that also allow

[13] http://www.mesa-imaging.ch
[14] Note that the benchmarks were conducted in early 2013. The particular versions of the libraries that were used are shown in the table
[15] Except for VxL, whose License is not publicly available

	OpenCV	**IPP**	**Halcon**	**Matlab**	**PCL**	**CImg**	**CCV**	**RAVL**	**VxL**	**ICL**
aspects regarding ease of use										
Languages	C, C++, Java, Python	C	C, C++, C#, VB,Delphi	matlab-script	C++	C++	C	C++	C++	C++
Distribution	src/bin.	bin.	bin.	bin	src/bin.	header	src	src	src	src/bin[14].
Platforms	all[1,17]	all	all	all	all	all	Linux[2]	Win./Lin.	Win./Lin.	all
License	BSD	comm.	comm.	comm.	BSD	CeCill	BSD-3-Cl.	LGPL	VxL-Lic.	LGPL
Documentation	A/M/T/B[3]	A[4]	M/T/A[4]	A/M/T/B	A/T	M/T	M	A/M	A/O/T	A/M/T
Version	2.3[15]	7.07	11 (demo)	R2013a	1.6	1.5.4	04/2013 (unstable)	1.1.2	1.14	8.1
Build System	CMake	–	–	–	CMake	–	make	automake	CMake	CMake
Hosted at	source forge	Intel ® website	MVTec website	MathWorks website	point-clouds[6]	source forge	source forge	source forge	source forge	CITEC[8] servers
framework functionality										
Supp. Functions	yes	no	yes	yes	yes	yes	no	yes	yes	yes
Control Strategy	yes	no	yes	yes	no	no	no	n/a	yes	yes
Build Tools	no	no	yes (IDE)	yes[13]	no	no	no	yes	yes	yes
speed										
Optimizations	all[5]	all	all	all	all	none	MT,SIMD	SIMD	SIMD	all
Sobel-X filter[10]	4.0ms	0.1ms	1.0 ms[7]	1.8ms	–	2.0ms	1.5ms	6.4ms	0.7ms	0.1ms
FFT	4.7ms	2.3ms	5.5ms	3.6ms	–	9.0ms	–	25ms	14ms	2.3ms
Threshold	0.03ms	0.03ms	0.3ms	17ms	–	0.2ms	–	5ms[12]	0.15ms	0.03ms
Local thresh.[11]	1.7ms	–	3.0 ms	–	–	–	–	–	–	1.0ms
Connect. Comp.	2.1ms	1.0ms[9]	n/a	4.6ms	–	5.4ms	–	15ms	2.6ms[16]	1.4ms

1) Windows Linux and MacOS **2)** perhaps others **3)** **A**PI, **M**anual, **T**utorial, **B**ook (printed), **O**nline book **4)** professional support provided **5)** SIMD, GPGPU, MT(multi-threading) **6)** www.pointclouds.org **7)** all Halcon benchmarks were measured in the HDevelop IDE **8)** https://opensource.cit-ec.de **9)** result is not a list of regions, but a region-image that needs further processing **10)** mask size 3, output depth 8 bit (if possible) **11)** mask size 30×30 **12)** manually implemented using pixel access operator **13)** Matlab provides a full-featured IDE for its own script language **14)** Binary deployment is planned **15)** The most recent version 3.x, which was released officially in June 2015 has identical features **16)** Some regions were missing **17)** Also an Android version is available

Table 3.1: General comparison of different computer-vision libraries. The vertical sections refer to the library requirements defined in section 3.1. The speed section also provides a minimal set of selected benchmarks, obtained on an Intel ®Xeon ®E5530 CPU, running at 2.4GHz, with 64bit Linux. For all tests, a gray scale 512×512 *lena* test image was used. These benchmarks are not intended to systematically compare the libraries, but to give a coarse insight of the overall performance reflected by the speed of common low- and intermediate-level procedures. All libraries were compiled with speed-optimization and linked in order to use all accelerating dependencies available, in particular ICL and OpenCV were linked against Intel ®IPP

for the implementation of proprietary software. Another important aspect that usually affects a programmer's decision about which library they will use, is the design of the interfaces and the quality of the documentation. These are explicitly excluded from the comparison because they are highly subjective. Even though the speed of the libraries cannot be evaluated in detail, a small set of exemplary functions were benchmarked. This gives, in addition to the set of used optimization paradigms, a coarse qualitative idea of the overall optimization status of the libraries.
Table 3.2 contains information about the *function volume* aspect of the libraries. The table lists a set of commonly used image processing functions, that are split into several sub-topics that reach from common classical low, intermediate, and high-level functions to 3D point cloud processing. It also lists support functions for graphical user interfaces, image I/O and mathematical tools, such as linear algebra and machine learning functions.

	OpenCV	**IPP**	**Halcon**	**Matlab**	**PCL**	**CImg**	**CCV**	**RAVL**	**VxL**	**ICL**
low-level operations										
Convolution	++	++	++	++	3D	++	++	++	++	++
Morphological Op.	++	++	++	++	3D	++		++	++	++
Color conversion	++	++	++	++	+	++	+○[7]	++	+	++
Affine Operations	++	++	++	++				+	++	++
Adaptive Threshold	++		++	+						++
FFT	++	++	++	++	++	++		++	++	++
intermediate-level operations										
Region analysis	++	+	++	++	++	+		+	++	++
Image features	++		n/a	++	2D/3D		++	+	+	++
Segmentation	++	+	++	++	2D/3D			+	++	++
high level operations										
Face tracker	++			+			++		++	○[4]
Person tracker							++			○[2]
6DOF Object tracker					++					○[1]
SLAM					++					○[1]
3D data and point cloud processing operations										
Point cloud proces.			++	+	++				+	++
Camera calibration	++		++	++					++	++
3D Segmentation			++		++					++
3D Shape Fitting			++		++					○[2,3]
3D Features			n/a		++					○[3]
Kinect Support	○[2]		++		++					++
graphical user interface and visualization support										
Image Visualization	++		++	++	++	++		++	++	++
Image Annotation	++		++	++	++	++		++	++	++
3D Visualization			++	+	++	++		++	++	++
GUI creation	++		++	++				++	○[6]	++
image I/O support										
File I/O	++		++	++	+	++	+	+	++	++
Camera I/O	++		++	++	+	++		+	++	++
Network I/O	++		++						++	++
built-in mathematical tools										
Linear Algebra	++	++	++	++	++[5]		++	++	++	++
Machine Learning	++	+	++	++	+			++	++	++

+): minimally supported **++)**: well supported ○): indirectly supported **1)** planned **2)** as external projects **3)** via PCL compatibility **4)** via OpenCV compatibility **5)** using the Eigen library **6)** provides interfaces to MFC, Qt and GTK **7)** only RGB to YUV is provided

Table 3.2: Comparison of image libraries by means of a selection of functionalities commonly used in the development of computer-vision applications. Empty cells mean that the corresponding functionality is not provided by the library, or that it could neither be found in the library documentation nor in the libraries source files. Externally available projects that use a library to provide a certain functionality are not taken into account if they are not part of the standard library distribution.

3.3 The Image Component Library (ICL)

The development of the Image Component Library (ICL) began in early 2006. At this point, the Neuroinformatics Group in Bielefeld University was trying to establish a new standard for computer-vision applications internally, in particular in order to facilitate robotics research. Since none of the existing libraries, not even OpenCV, were completely satisfactory, it was decided to implement a wrapper around an already established library. At this time, the current OpenCV version was 0.8, and it mainly provided fallback implementations for the non-free Intel ®IPP. So rather than *wrapping a wrapper* of Intel®IPP, ICL was directly developed on top of Intel®IPP, however, in contrast to OpenCV at that time, which was written in C, ICL was written in C++.
In 2011, ICL was officially released as an open-source library and its license was adapted form GPL to LGPL. Since that time, ICL has been officially accessible as an open-source library via CITEC's open-source server[16]. Its name was originally chosen to reflect the internal structure of the fundamental image class, which supports the construction of images from different shared-channels – image components.

3.3.1 Design Principles

ICL was intended to be well suited for both programming beginners, but also advanced programmers. ICL's main design principles reflect the general requirements for computer-vision libraries defined in section 3.1.

ICL is a Framework

ICL was designed to provide everything necessary to implement complex and even interactive computer-vision applications. This includes not only computer-vision related functions, but also a large set of utility functions and classes, such as program argument evaluation, matrix classes and an easy-to-use GUI creation framework. In addition, ICL provides a control strategy that elegantly allows the creation of multi-threaded GUI-applications. A dedicated thread processes user-interface events while simultaneously performing image processing in one or more working threads. Moreover, ICL provides ready-to-use build tools, that can be used to compile simple programs linking against ICL directly and to create a source-, documentation- and makefile-structure for smaller projects.

Optimal Performance

ICL provides highly optimized computer-vision related functions. The exemplary benchmarks given in table 3.1 show that ICL can even rival OpenCV and Halcon. This is partly achieved by wrapping Intel ®IPP functions, professionally tuned using SIMD instructions and multi-threading. If compiled without Intel®IPP support, C++-fallback implementations are provided.
The more common a specific function is, the more time was spent manually tuning it. Some of these implementations are even faster than the corresponding IPP functions. For both SIMD- and GPGPU-based optimization, support classes are provided that also allow ICL-users to speed up their implementations more easily. Furthermore, all classes are set up to provide and use internal image buffers to avoid time consuming memory allocations at run-time.
In addition to the programmatic optimizations that are carried out, the class interfaces were tuned

[16] http://opensource.cit-ec.de/projects/icl

in order to provide fast interfaces for custom implementations. The image class provides direct access to internal image data, where each channel is always packed in a single data segment. Furthermore, images can be shallowly *wrapped around* existing data segments, to avoid expensive element-wise data copies. That is, while the small memory-footprinted image meta-data, such as size, timestamp or pixel format are copied deeply, the actual pixel-data, whose memory-footprint is commonly in the order of megabytes is *imported* into the image by simple pointer-copies.

Extensible and Convenient Interfaces

Another design principle was to provide powerful and extensible generic interfaces that are not only used within the library, but also provided as tools that can be used for external applications. For example, the image acquisition framework provides a plugin interface class that is not only implemented for a large set of image sources, such as image files, many camera interfaces, video files and network streams, but which can easily be augmented with custom image sources. An overarching manager class allows for a very simple and intuitive selection of an application's image source using command line parameters. For optimal generality, often string-based interfaces are used in non-time-critical functions.

Simple and easy-to-use C++ interface

ICL's API is designed to allow for writing short, but readable and intuitive code. In particular, ICL tries to reduce the amount of *boiler plate code* necessary to implement even interactive applications. Implementation details are often explicitly hidden using C++'s *private implementation* (pimpl) pattern, which helps not only to reduce the complexity of the class interfaces, but also to reduce the overall compilation time when linking against ICL. Almost all external dependencies are wrapped completely, allowing ICL to provide more consistent interfaces. In contrast to many other libraries, no external utility libraries such as *Eigen* for matrix algebra or *boost* for general purpose tools must be learned. Only if access to the underlying implementation details is necessary, is it explicitly provided. Together with shallow inheritance trees, as well as consistent naming, coding and documentation conventions, ICL is optimally suited for both programming beginners and experts. Finally, ICL has no compulsory library dependencies, which significantly simplifies the install process.

3.3.2 ICL Modules

ICL is subdivided into ten libraries, arranged in a linear dependency graph (see the first column of Table 3.3 for its structure). Each library contains a module that provides a specific function set. On the lowest layer, the ***utils*** module defines general support classes, types and functions. These are not directly related to computer-vision. Also, not exclusive to computer-vision tasks are the contents of the ***math*** module, which provides mathematics related classes for linear algebra, machine learning, optimization and neural networks. The ***core*** module contains ICL's main image classes and provides a set of fundamental image processing features, such as type and color conversions. Low-Level filters are located in the ***filter*** module, which distinguishes between unary and binary operators, depending on the number of input images. The ***io*** module contains the image input and output framework. Intermediate-level image processing functions are located in the ***cv*** module. On top of this, the ***qt*** module introduces ICL's GUI creation framework including hardware-accelerated 2D/3D data and image visualization and annotation. The ***geom*** module provides tools and functions for 3D computer-vision, such as camera geometry and calibration and point cloud

Module	Description	Main contribution (classes/functions)
utils	general purpose utilities	Basic types, program argument evaluation, *Configurable* interface, time-related classes exception-types, configuration files, multi-threading tools, string manipulation tools, generic *Function* class, random number generators, support macros
math	mathematics and machine learning tools	Dynamic- and fixed size matrix and vector classes, levenberg-marquard optimization, FFT-framework, RANSAC-optimization, simplex optimization, stochastic optimization, vector quantisation, SOM and LLM networks, quad-tree and octree classes, least-square-based model fitting, generic polynomial regression network
core	basic image processing types and functions	Main image classes *ImgBase* and *Img*: data handling, shallow copies, pixel access, ROI-handling, channel management, copying, converting, scaling, minimal statistics, color format conversion
filter	image filtering framework	Unary operators: affine operations and warping operations, convolution, morphology operators, wiener filter, gabor filter, median lookup operations, color-segmentation, canny edge detector, champfering operator, FFT-operators, integral-image, (local/global) threshold, arithmetical operations, logical operations, channel mangling, gradient image, rectification. Binary operators: arithmetical operations, logical operations, pixel-wise comparison, proximity measurement
io	image input/ output framework	Grabber-framework for image input with back-ends for: FireWire, Video for Linux, image files, video-files, Microsoft Kinect, Gig-E cameras, shared-memory-based streaming, network-streams using RSB[Wienke and Wrede, 2011], adaptors for OpenNI and OpenCV. Image output framework with back-ends for: image-/ video-files shared-memory and RSB-based network streams. I/O support classes for image compression
cv	intermediate level computer-vision utilities	Connected-Component framework, SURF-feature detection framework, blob-searching and tracking, flood-filling, Hough-line detector, generic tracking framework, mean-shift tracking, template matching and tracking
qt	GUI-creation-tools image/data visualization	GUI-creation framework (including all common UI-components), image visualization 2D/3D image annotation framework, function/data-plotting, mouse-/keyboard handlers
geom	3D geometry tools and algorithms	Camera class, camera calibration, single/multi-view geometry, 3D scene graph for OpenGL-based 3D visualization on top of common 2D images, point-cloud-processing framework, including PCL compatibility layer, RGBD-image grabber framework, RGBD-mapping, automatic 3D point cloud segmentation
markers	fiducial marker tracking framework	Generic plug-in-based framework for common fiducial marker types: ARToolKit, BCH-code, Amoeba and others (see Sec. 4.1.1), single/multi-view 2D and 6D marker pose-estimation
physics	shallow bullet physics engine wrapper library	Simple and easy to use wrapper classes around most commonly used Bullet classes. Rigid objects, soft-body objects, constraints, motors, physics-world, seamless integration with ICL's 3D visualization framework, modeling of paper

Table 3.3: Compact overview of the contents of ICL's modules. Each module uses functionality from the preceding modules, leading to a linear dependency graph. The only exception to this is the **physics**-module, which depends on the **geom**-module, but not the **markers**-module.

processing. It also extends the visualization engine from the ***qt*** module by a scene-graph-based 3D visualization toolbox. Using functions from all other modules, the ***markers*** module provides a generic fiducial marker detection framework. The lastly added ***physics*** module that basically emerged from the work presented in the thesis, provides a shallow wrapper and integration of the Bullet physics engine[bul]. A concise overview of the contents of the different ICL's modules is given in Table 3.3.

3.3.3 Documentation

As discussed in Section 3.1, a very important feature of software libraries is how well they are documented. ICL provides different levels of documentation, each optimized for a special purpose. The starting point is ICL's website[17]. From here, users can get information about downloading and installing ICL instructions, as well as tutorials, the ICL manual and ICL's API documentation. In contrast to many other libraries, ICL's API is completely documented and includes code samples, benchmarks and formulas. However, it turned out that the API documentation is usually more

[17] www.iclcv.org

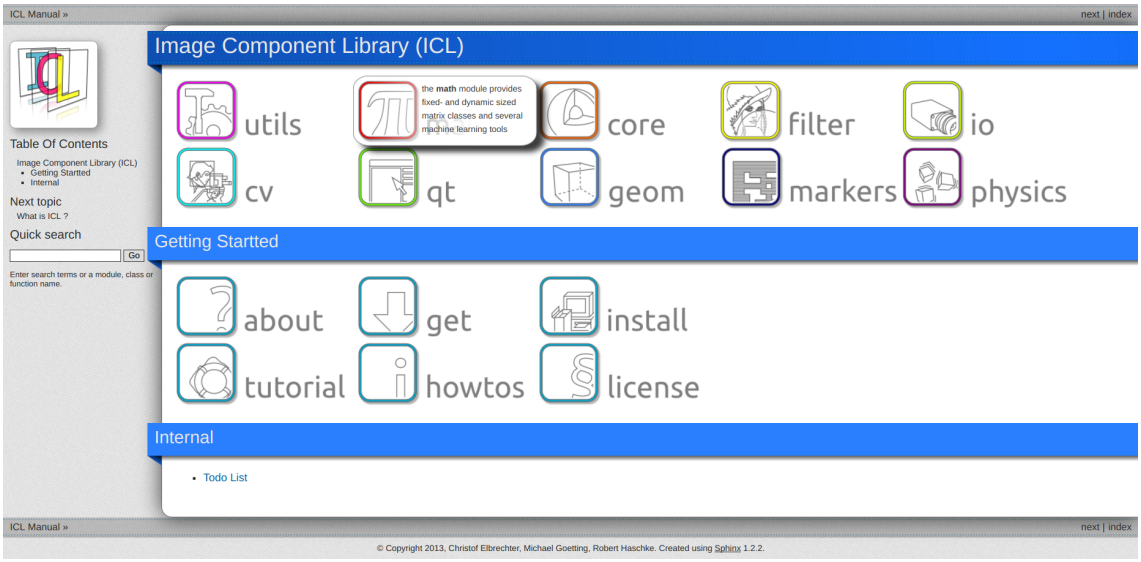


Figure 3.1: Screenshot of ICL's manual which can be found at www.iclcv.org. Each module has its own subpage on which all relevant concepts and classes are presented. In addition, information is given about how to download and install ICL. The manual is directly linked to ICL's API documentation.

helpful for users who search for some specific functionality. In order to provide a better overview of the contents of ICL's modules, the *manual* is provided, which is written using the sphinx documentation generator[18]. It reflects ICL's modular structure by providing a separate section for each module. By using a sphinx-extension that allows for linking documentation written in sphinx with the API documentation, the user can directly jump from a class mentioned in the manual to its more detailed description in the API. Furthermore, a custom-written java-script-based extension creates tool-tips for classes, functions and types, providing direct information without having to follow a link. www.iclcv.org directly shows the manual's front page (see Figure 3.1).

3.3.4 Positioning ICL in the Landscape of Vision-Libraries

ICL can best be compared with OpenCV, RAVL and VxL. Proprietary libraries, such as Intel®IPP and Halcon cost money for each license, and are therefore less likely to be used in research facilities and particularly not by students. While a license for Intel®IPP is still affordable (non-commercial use: ≈100 EUR, commercial use: ≈200 EUR), a single Halcon license costs about 3000 EUR for research purposes (more than 6500 EUR for profit-oriented use). While its speed is outstanding, Intel®IPP only provides low and a few intermediate-level functions and it does not provide any support functions to acquire or to visualize images, which implies that other libraries need to be used to achieve this.
The price for the Matlab image processing toolbox is slightly more expensive than Intel®IPP (single academic license for Matlab: 300 EUR and an additional 200 EUR for each toolbox), however many research institutes have concurrent licenses that can be used by many developers at once. Due to its lack of speed, Matlab is not very well suited for real-time applications. In particular, custom-written algorithms are only real-time-capable if they are implemented in C and linked using Matlab's native interface. However, in this case the advantages of using an intuitive scripting language diminish and the distribution of the code (made up of both, Matlab script and C source code) complicates development and also prolongs development cycles.

[18] http://sphinx-doc.org

The CImg library is well suited for first steps into the computer-vision domain. Its interfaces are simple and do not even require explicit buffer management since functions usually return newly instantiated result images. While this can be very helpful for beginners, it also leads to major performance issues due to extensive memory allocation and deallocation at run-time, which can be seen in the benchmark results in Table 3.1. Even though the motivation of shipping the CImg library as a single header file is somewhat comprehensible, it is not optimal. It does not only defy all coding practice standards, but also leads to extremely long compilation times.
The cache architecture of the CCV library is an interesting feature, but whose advantages are however not completely obvious. Furthermore, CCV is not very well suited for application development, because the provided set of low and intermediate functions is very small. Even though it is written in C, its interfaces are very clear and consistent, but in the opinion of the author, the documentation needs to be improved.
In contrast to the previously mentioned libraries, the PCL library is highly optimized for 3D point cloud processing, where it has already advanced to a quasi standard in the open source domain. PCL is intrinsically linked with Willow Garage's OpenNI framework, which provides a large set of high-level tools based on 3D-sensing. PCL is well documented and its interfaces are written in modern template-based C++. The exhaustive use of C++-templates however makes some of the class interfaces very complex, which can lead to a steep learning curve for programming beginners.
The RAVL library is a full featured computer-vision framework, including support functions for creating interactive GUI-based applications. However, it mainly concentrates on providing low-level functions. It also provides a very large set of low-level support data types. RAVL is well documented using its own code-based documentation generator, however the documentation lacks examples, which would make an entry significantly easier to understand. Its template-based interfaces are very consistent, but also complex and often use cryptic class names.
VxL is spread over a set of different libraries with partly unintuitive names, such as *vil* and *vil1*. It provides a large set of support functions, including an optional reimplementation of C++'s standard-template-library (STL), which is intended to be more portable to the different supported platforms. It's STL-style interfaces are very readable and consistent. A free online-book, with many examples is a great help for both beginners and advanced programmers. However, the extensive use of hierarchically nested C++-templates, which is even necessary for simple examples, makes its use difficult for non C++ experts.
OpenCV, the standard library for open-source computer-vision has been significantly improved since its development was taken over by Willow Garage. New and more intuitive library naming conventions, a C++ interface and Python wrappers facilitate its use. The documentation is outstanding with many examples and even printed books are available. The major change in how OpenCV should be used is driven by the new C++-interface. It solves memory management, significantly helps to reduce the amount of *boilerplate code* and also brings new features such as command line parsing and versatile GUI creation. However, the existence of two common interfaces (C and C++) has its drawbacks. While Willow Garage has tried to establish the C++-interface as a new standard, it does still not cover the whole range of functions that are available in the C-interface. Furthermore, many examples or tutorials that can be found in Internet forums still use the C-interface.
ICL justifies its existence by trying to avoid the pitfalls some of the other libraries have. It was directly developed in C++, so there is no legacy interface that needs to be supported. Its interfaces are kept as simple as possible and C++-templates are only used when completely necessary. ICL's abstract image base class ensures that templates can be avoided as long no manual pixel access is necessary. ICL provides full API documentation including many examples, a step-by-step tutorial and also a system manual. A lot of effort was put into light-weight interfaces that allow for reducing the amount of boilerplate code, necessary, even for interactive applications, and ICL provides a large set of support functions and classes that significantly facilitate application development. It

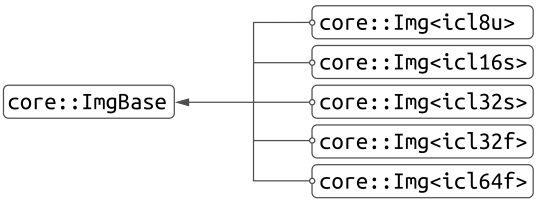


Figure 3.2: Inheritance diagram for ICL's image classes. The *ImgBase*-class provides all image information except for the actual image pixel data. The actual data type (e.g. 8bit unsigned integer *icl8u*) is defined by the template parameter of the derived $Img\langle T\rangle$ class.

is self-contained and even provides functions for 3D point cloud processing, which has recently become a standard field in computer-vision.
However, the existence of OpenCV and the PCL library can not be ignored. Therefore ICL provides fast conversion functions from and to OpenCV's image types, which allows for a seamless integration with OpenCV-based code. For point cloud processing, a generalized point-cloud class interface was developed, which is implemented not only by ICL's point-cloud type, but also by a PCL-point cloud wrapper, allowing for the direct use of PCL algorithms.

3.4 Important Tools for this Work

In this section, some selected aspects of ICL are presented more detailed. The selection is driven by the relevance of each aspect to research done for this thesis. The development of ICL was mainly driven by the idea of providing a versatile computer-vision library that enables researchers to easily develop even complex applications. In the beginning of the development process, most of the effort went into the development of a fast and yet easy-to-handle image class and a large set of low-level image processing functions. Most of these were accelerated using optional Intel®IPP back-ends. Once this was completed, other modules were successively added.
Most of the functionality necessary for the desired robotic paper manipulation system was first separately developed and integrated into a working prototype. These tools were then usually re-implemented in a more generic manner, no longer specialized for the specific task. If possible, the more generic framework was then included into ICL in order to extend its function volume. This approach does not only allow for re-using the functions in other projects directly, but it also significantly helped to optimally structure the source code.

3.4.1 Easy to Use Core Functionality

The core functionality of an image processing library has consequences for everything else that is implemented in the library. Important basic concepts, such as image classes and filters, are frequently used and therefore have a high impact not only on the simplicity to implement custom functions and applications, but also on the readability of the produced source code. A couple of representative ICL fundamentals, namely the image class hierarchy and the filter concept, are now presented.

The Image Classes

The most commonly used type in a computer-vision library is most likely the one that represents an image. Often contrary requirements lead to either fast, but hard to use, interfaces, or slow, but

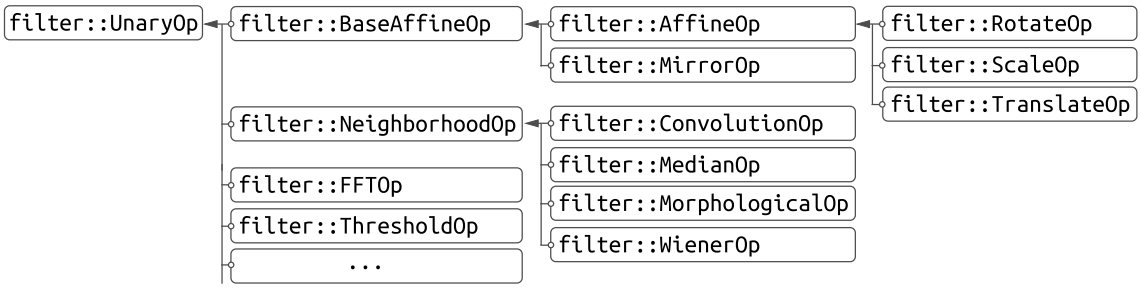


Figure 3.3: Inheritance structure of ICL's main image filter class *UnaryOp*. All subclasses obey the same interface, allowing for easy replacing and stacking.

easy to use ones. In C++, many libraries (e.g., VxL, RAVL, CImg, PCL) use a class template (e.g., *Image*⟨*T*⟩) to represent images with different pixel types. While this helps to significantly reduce the size of the implementation and helps to avoid issues when accidentally misinterpreting *int* pixels as *float* pixels, it also has some drawbacks for other library interfaces, as well as for the programmer, who uses instances of that class. In C++, a template instance of type *Image*⟨*int*⟩ and *Image*⟨*float*⟩ have little in common, except for their identical structure. Therefore generic functions, able to work on different versions of the image class must also be implemented as templates. This not only increases compilation times, but it also complicates all interfaces in general, which can be especially problematic for beginner programmers. In C, different pixel types can be represented by *void*-pointers or using a *union* structure (e.g., in OpenCV's C-interface). In both cases, the actual type of the data must be encoded explicitly, i.e., not by the language's built-in type mechanism, leaving the work of type checking and switching to the programmer. This is particularly important, since type-misinterpretation can not be detected by the compiler. The *Mat*-class in OpenCV's C++-interface is not implemented as a template. Instead a run-time parameter is used to determine an image's pixel type, which basically complies with the C-interface.
ICL uses a combination of a normal and a template class to circumvent these issues. The abstract *ImgBase* class manages all image parameters that do not depend on actual image pixel data, such as size, channel count or region of interest. In addition, it holds a run-time *depth* parameter that defines its actual type, which is a specific version of the *Img*⟨*T*⟩-template (see Figure 3.2). This structure allows for providing general, but template-less interfaces using the *ImgBase*-class. It is only when pixel-access is needed that the *ImgBase*-interface is safely down-cast into its actual type.

Filters

Even though filters are a very common concept in image processing, there is no unique general definition available. Sonka et al. [2007] even avoid using the term *filter*. Instead, the term *image pre-processing* is used, to generally describe operations, that have intensity images for both input and output. Furthermore, they subdivide the set of pre-processing operations into four classes, according to the amount of neighboring pixels used to compute a single output pixel.
ICL adopts this definition, but also distinguishes between *unary*- and *binary* operations, according to the number of input images used. The set of binary operations, represented by the *BinaryOp* interface, is again subdivided into pixel-wise comparison, arithmetical and logical combination and proximity measurement of two images. The set of unary operations extending the *UnaryOp* interface is much larger and hierarchically structured (see Figure 3.3). The *UnaryOp*-interface defines generic *apply*-methods that compute an output image given an input image. This allows for using all unary operators in exactly the same way, enabling the programmer to generically

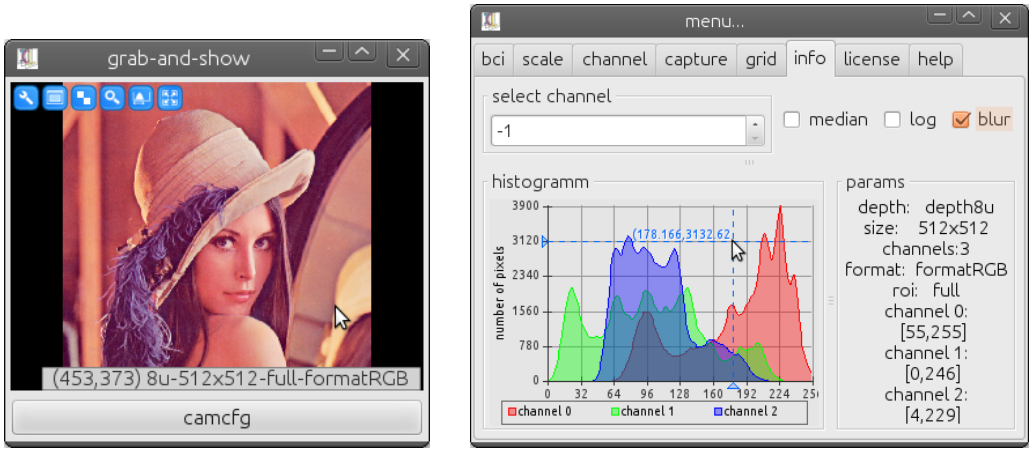
```
#include <ICLQt/Common.h>

GUI gui;                          // global GUI instance
GenericGrabber grabber;           // image source

void init(){
   grabber.init(pa("-i"));        // init from prog. arg.

   gui << Image().handle("img") // add image display component
       << CamCfg()              // add camera configuration comp.
       << Show();               // create and visualize gui
}

void run(){
   gui["img"] = grabber.grab(); // grab image and visualize it
}

int main(int n, char **args){    // tie everything together
   return ICLApp(n,args,"-input|-i(2)",init,run).exec();
}
```

Listing 3.1: C++ source code of an example for generic image acquisition and visualization.

exchange or to *stack* unary operators if necessary. The explicit inheritance structure also implicitly helps beginners in the image processing domain to classify or to search for unknown operations.

3.4.2 Grabber Framework for Dynamic Image Source Selection

Driven by the goal of providing a built-in system that allows for an easy implementation of applications that can deal with arbitrary supported image sources, a grabber-framework was developed. It uses string-based interfaces for both device selection and configuration. In many applications, an image source is initially selected, which is then used at run-time to acquire new images. Here, usually the programmer either has to decide on a specific input type or an input device selection mechanism (e.g., via program arguments) must be implemented manually. Such a mechanism, however, enables not only the user, but also the developer to easily switch between processing real-time camera images, a list of image files or even offscreen-rendered images of dynamic artificial scenes received via network or shared memory.
In ICL, a powerful *grabber*-framework is provided that defines an easy-to-use interface to select a supported input device. The selection mechanism is usually bound to program arguments, to manually select an application's image source. To this end, a plugin-system was developed, which maintains all available backends in order to provide a slim and simple-to-use interface called *GenericGrabber*. An instance of this class can easily be initialized with any of the supported backends. It can be configured using automatically created configuration files as well as by optional program options and it supports the automatic creation of a GUI-component that allows the user to interactively adapt device properties at run-time. Listing 3.1 shows a minimal skeleton for a simple *grab and show* application. A screen-shot of the resulting application is given in Figure 3.4.
The resulting application uses a single program argument *-input* (alternatively *-i*), which expects two sub arguments that are used to pick an image source backend and to select a backend-specific device. As shown in the example screenshot (see Figure 3.4a), *-i create lena* selects the *create* plugin, which simply creates a common test image, specified by the second sub-argument. *-i dc 0* would select the *dc* plugin, used to grab from the first fire-wire camera device found and *-i file 'images/*.jpeg'* would grab all jpeg files in the *images* directory successively. The created appli-



(a) (b)

Figure 3.4: **(a)** Screenshot of resulting application (see Listing 3.1), when started with input selection *-i create lena* **(b)** External widget menu providing extra image information and visualization options.

cation also provides a facility to get a list of supported devices, using *-i list all*. Furthermore, the same interface can also be used to set device properties, e.g. *-i dc 0@gain=500* would initially set the selected dc camera's *gain* property to 500. Supported properties and allowed property values can also directly be queried from command line, e.g. with *-i dc 0@info*. The created *CamCfg* GUI component (see Listing 3.1), displayed when pressing the *camcfg*-button (see Figure 3.4a) automatically creates control interfaces for all initialized *GenericGrabber* instances.
ICL provides a similar but slightly less complex framework for image outputs, with many supported backends such as writing image files, video files and streaming images via network or shared memory. Furthermore, each image display component provides a menu (see Figure 3.4b), allowing – amongst other things – images that are displayed to be captured automatically. This can be used to capture a video of a display, or to use the output images of one application as input for other applications.

3.4.3 2D and 3D Visualization

Another important feature, missing in most other libraries, was fast and intuitive image visualization and annotation. Other libraries often perform image annotation directly in the image space, which means that primitives, such as text-labels, are directly rendered into the image pixel matrix. This is not only suboptimal due to the fact that it can usually not be accelerated by hardware rendering, but also by the fact that the resolution of the annotation directly depends on the image resolution. In particular when working with small images, more complex annotations, such as symbols or text, can become very pixelated. This can only be avoided by explicitly scaling up the image, which entails having to also transform all annotations to the new size.

Visualization Framework

For ICL, a new visualization framework based on OpenGL was developed, which exploits hardware acceleration for image-rendering and 2D/3D annotation. The framework is basically split into three layers. The bottom layer is defined by the *ICLWidget* component, which provides func-

tionality for hardware accelerated image visualization. It displays images as OpenGL textures, allowing for hardware accelerated scaling and zooming and for fast brightness and contrast adjustment. It also automatically scales the visualized image to use the widget's space, while preserving its aspect ratio and it seamlessly solves threading issues caused by image updates posted from outside the application's GUI thread. Furthermore, it provides an easy-to-use interface for installing mouse-event handlers. In contrast to Qt's mouse-event mechanism, ICL's mouse events provide information (e.g. color or position) about the *clicked* image pixel, automatically taking into account current image scaling and zooming. The *ICLWidget* component is also endowed with two kinds of menus, an *on-screen-menu* and an external one. The on-screen-menu, is defined by a set of blue buttons at the top edge, visible only when the mouse is over the display component (see Figure 3.4). It provides direct access to the most common functions, such as zooming in, entering full-screen mode or displaying the external menu. The on-screen-menu can also be customly extended in a programmatical manner. The external menu (see Figure 3.4b) is arranged in several tabs providing visualization-related as well as more general controls and information.
Extending the *ICLWidget*, the intermediate layer is defined by the *ICLDrawWidget*, which additionally provides a state-machine-like image 2D-annotation interface. It accepts drawing commands that are, similar to the handing of mouse events, parametrized in units of image pixel coordinates. All visualized primitives, such as lines, polygons and text are automatically aligned with the background image and then efficiently rendered using OpenGL.
The top-most layer, given by the *ICLDrawWidget3D* class, extends the visualization mechanism by an extra interface to also render 3D primitives on top of the image. However, unlike the state-machine-like interface of the 2D drawing component, a more sophisticated toolbox for 3D visualization is provided. Basically, the *ICLDrawWidget3D* can be used to link native OpenGL code into the rendering loop of the component, that is executed after rendering the background image. However, it turns out that this usually leaves too much work to the programmer. In particular adjusting OpenGL's camera parameters in such a way that real object and virtually rendered object actually align in the final image, even when zooming in, or adapting the widget size is a non trivial task.

Slim Scene Graph For 3D Overlays

At this point the integration with the *Scene*-class, provided by the *geom*-module, can easily help to save many hours of work. *Scene* instances are simply filled with objects that can be structured in an object tree, providing a relative transformation at each node. Once objects are added, one or several *Camera* instances are needed that define *how* or *from where* the scene is to be rendered. Cameras can either be defined manually or their extrinsic and intrinsic parameters can be easily estimated using ICL's built-in camera calibration tool in combination with a calibration object (see Figure 3.5). For each registered camera, the scene provides an OpenGL callback, that can directly be linked to an *ICLDrawWidget3D*'s rendering loop.
The 2D and 3D visualization framework was one of the most helpful support tools for the development of robotics applications because it allows for quickly adding debug visualization, which is often of great help when working with physics engines. The 3D visualization engine is highly flexible, ranging from being able to add simple geometric primitives such as boxes or spheres with a single command to registering complex native OpenGL callbacks and even defining vertex and fragment shader programs.

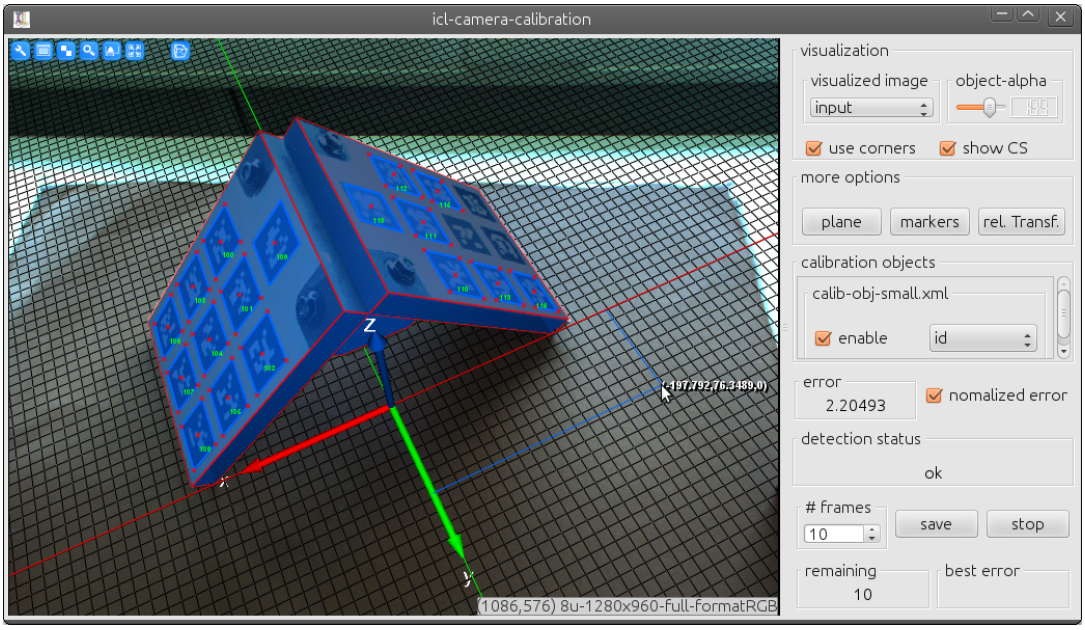


Figure 3.5: Screenshot of ICL's camera calibration toolkit. The calibration is performed using a 3D calibration object, whose geometry is described in an XML-file. The object is endowed with fiducial markers. The marker's detected centers and corners are used in the calibration process, which is performed in real-time.

3.4.4 Marker Detection Toolbox

In order to perform paper detection, different fiducial marker tracking libraries (see Section 4.1.1) were examined. However, it turned out that these were either not freely available, their license was not compatible with ICL's LGPL-license or they were simply not fast or accurate enough. Furthermore, most available libraries were difficult to use or to integrate. Therefore it was decided to endow ICL with its own marker detection framework.

As a first step, a new marker layout was designed in order to combine fast detection with 6D pose detection capabilities. Even though these markers were successfully used in a deformable paper tracking system (see Section 4.2), it turned out that in the desired configuration, defined by image resolution, physical marker size and average distance of markers to the camera, many markers were either not detected, or worse, their ID was extracted wrongly. Therefore, once again, alternative marker types were reviewed, leading to the idea of implementing a built-in framework from scratch, able to track all common marker types. This would not only allow for finding the best suited maker-type for the next version of the desired paper-detection system, but also be a valuable feature building block for ICL.

The public interface of the plugin-based framework basically consists of two classes. The *FiducialDetector* can be set up to use one of the existing marker-type-related backends. After marker detection, it returns a list of *Fiducial* instances, each providing all available information for a detected marker. The *Fiducial* interface is generic and works for all implemented backends. Internally, marker features, such as the marker boundary or the 6D pose of a marker, are estimated in a deferred manner, meaning that features are only computed when the corresponding getter method is actually called. Features once computed are automatically cached to speed up successive calls.

A comparison of the different fiducial marker types and possible tracking libraries including ICL's marker tracking framework is given in Section 4.1.1. The marker detection framework is also used for ICL's camera calibration toolbox (see Figure 3.5).

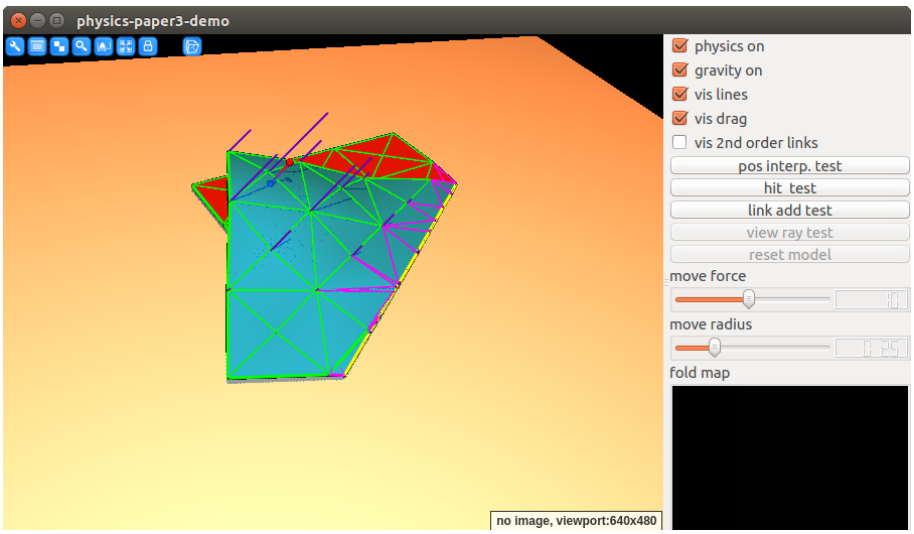


Figure 3.6: Screenshot of the interactive physical paper model editor. In the editor, the can be manipulated in a mouse-based drag and drop manner, the possible interactions include dragging parts of the model in 3D space, adding fold lines and memorizing the deformation along fold lines. Internally, the model is moved using the model control law presented in Section 6.1

3.4.5 Soft and Rigid-Body Physics Module

Due to the physical paper models that were needed by the robotic systems to pick up (see Chapter 4) and to fold (see Chapter 5) paper, a connection to the Bullet physics engine was required[19]. The engine had to be integrated with ICL's *geom* module to exchange information about geometry and other physical quantities between the different domains that were covered by the final demonstrator systems. In particular a seamless unification of the different coordinate frames for vision, physics, (debug-)visualization and for the robotics system was necessary to achieve well-defined interfaces that allowed data to be exchanged intuitively and efficiently. A particular difficulty here were the different size units that were employed in the different domains. While ICL's *geom*-package assumes *millimeters* by default, the robotics framework uses *centimeters*. The integration with the Bullet physics engine was even more complex in this regard, as its internal solving mechanism is optimized to work optimally with object sizes in the order of a unit-less length of 1. Thus an internal scaling-factor was introduced, which had to be applied in a linear fashion to lengths but non-linearly to some other physical quantities.
A seamless integration was achieved by providing shallow wrapper classes for most of the common physics types provided by Bullet. All object classes are explicitly derived from the *geom*-module's *SceneObject* class, which allows for a real-time visualization of a dynamic physics scene to augment real camera images. A special *PhysicsScene*-class that uses multiple inheritance to extend both the *geom*-module's *Scene* as well as the *physics*-module's *PhysicsWorld* can be used to set up powerful physical simulations and a corresponding visualization with just a few lines of code. The most important feature for this thesis was the *SoftObject* class that, by wrapping around Bullets soft-body physics module, defines the basis for the different physical paper models that were developed. For the model that was used for robotic folding (see Section 5.3) and also for the final model presented in Section 6.1, an editor was created that allows the user to move a paper model and to dynamically add folds in a drag and drop manner (see Figure 3.6).

[19] The author thanks **Matthias Esau** for his help with physics module.

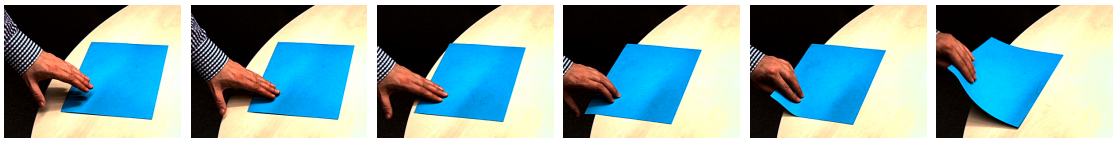
3.5 Discussion and Next Steps

ICL has become a powerful library and it has proven its versatility in a large number of projects. Its main competitor is certainly OpenCV, which has about 50 thousands users and is actively developed by more than 100 people. By providing an easy-to-use compatibility layer that allows image data-types to be exchanged with OpenCV in a very seamless manner, OpenCV algorithms and features can be directly included into ICL programs with only a few lines of additional code.
A very important next step will be to further work on ICL's community. Transitioning ICL to an open community project hosted on GitHub [20] was carried out in early 2018[21]. This will allow other people to contribute to ICL, to speed up the addition of new features and the fixing of bugs. Another important task for the near future will be to provide for integration with the robot control toolkit/platform ROS[22], in order to make it easier to use ICL when implementing ROS-nodes and to connect ICL-programs through ROS-data streams.

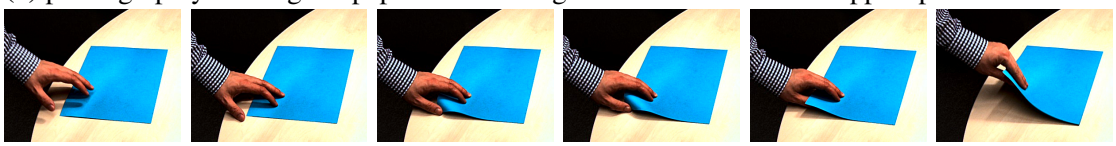
[20] www.github.com
[21] The author thanks Alexander Neumann for taking the leadership for this endeavor
[22] www.ros.org

4 Picking up Paper

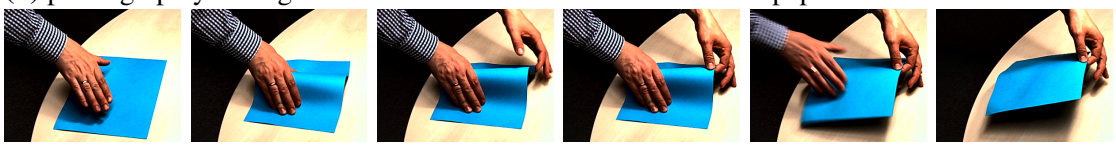
After investigating paper shifting and rotation in the 2D plane, the next logical step was manipulation in 3D. The first task was to pick up a sheet of paper that was lying on a flat surface. This allowed for the introduction and testing of a whole new processing pipeline, consisting of 3D detection, modeling of the deformation of the sheet of paper and new building blocks for robot control. The ability to pick up a sheet of paper is also a common prerequisite for many other more complex interaction sequences, such as moving paper in 3D or putting it into a drawer or a binder.



(a) picking up by shifting the paper over the edge of the table or other support plane



(b) picking up by slicing the thumb under a corner of the sheet of paper



(c) bi-manually picking up by bulging up the sheet of paper to allow for pinch grasping

Figure 4.1: Different common strategies used by humans to pick up a sheet of paper lying on a flat surface. **(a)** This technique allows the paper to be picked up without *damaging* it, but it assumes a table edge to be within reach. **(b)** The method shown here is usually the simplest way, but it can only be attempted if one edge of the paper is either already bent upwards or a slight fold at the corner of the paper can be risked. **(c)** This technique is only rarely used because bi-manual picking-up is *energy-wise* less efficient. If no table edge is within reach, it might be preferred over (b) to avoid damaging a paper's corner.

Even though picking up a sheet of paper is a very easy task for humans that is usually learned within a child's first year of life, it had not been solved for anthropomorphic robot hands. While in production facilities, picking up objects that do not allow for positive fitting grasps is usually performed using – depending on the material – either magnets or vacuum grippers, the use of anthropomorphic robot hands requires more human-like grasping strategies. Humans would usually either try to shift the sheet of paper towards an edge of the supporting surface to easily grasp it (see Figure 4.1a) or bring one finger under a corner of the paper sheet (see Figure 4.1b). The chosen technique depends not only on the paper-to-table configuration, but also on several external variables, such as obstacles on the table, the human-to-table position and orientation, whether a slight crease at the corner of the paper can be risked and on personal preferences. Seeing the human planning procedure as an energy minimization process, an edge that lies not perfectly flat on the support plane provides a good grasping affordance, which usually allows for an energy-

wise cheaper pinch-grasping of the corner. In contrast to this, if one has to pick-up an important document that must not be folded or even bent too much, shifting the paper to an edge becomes much more likely even if potential obstacles need to be avoided or displaced before. These two examples demonstrate that even supposedly simple tasks like this can not be planned properly without taking into account world knowledge.

Picking up Paper with the Robot

In order to be able to concentrate more on the low-level processing necessary to endow a bi-manual anthropomorphic robot system with the ability to pick up a sheet of paper lying on a flat surface, a grasping strategy was pre-defined. Initial experiments showed that pinch-grasping a single corner of the paper was not possible without the risk of damaging the robot hand. The main reason for this was the weak quality of tactile feedback provided by the fingertip sensors, in particular the missing feedback on the sides and the tops of the fingertips. In addition, the sheet of paper itself produces very little force resistance, and the forces produced cannot be measured with the current fingertip sensors. The shifting-technique, frequently used by humans, would basically be identical to the 2D paper shifting from the previous experiment (see Section 2.2) followed by a simple grasp. Therefore, the picking up sequence depicted in Figure 4.1c, which needs bi-manual interaction, was developed. The new sequence basically consists of two steps. First, the right hand bulges up the sheet of paper in the middle. Then, the left hand can be used to simply pinch-grasp the center of the bulge. Bulging the paper introduced the necessity to model the deformation of a sheet of paper. For this, two different approaches were developed, implemented and evaluated. First, a less complex, purely mathematical, model was used, which represented the sheet of paper as set of localized paper-patches each modeled by a 2D polynomial embedded in 3D space. In order to obtain a smooth surface the manifold is smoothed using soft-max interpolation. The second, more sophisticated, approach taken was a physics-based model implemented using the soft-body module of the open source Bullet physics engine[1]. For the perception of the paper and the tracking of its deformation, a multi-view camera setup was employed. In order to simplify the detection module, the paper was densely covered with fiducial markers on both sides. At this point, it turned out that existing, freely available, fiducial marker tracking libraries were not fast- or accurate enough. Therefore, a new fiducial marker layout was designed and a very fast detection module was implemented, which allowed the processing of mega-pixel image streams at up to 100 Hz.
This chapter is organized as follows. Related work is presented in Section 4.1 and then in Section 4.2 the developed fiducial marker tracking system based on fast connected component analysis in binary images is presented. The two modeling approaches are then introduced (see Section 4.3) and their accuracy is compared on the basis of real-world and artificial ground-truth data in Section 4.4. Section 4.5 will then present the robot control system that was implemented to pick-up a sheet of paper before the results are summarized and discussed in Section 4.6.

The work presented in this chapter is mainly based on the author's conference paper:

> **Bi-manual robotic paper manipulation based on real-time marker tracking and physical modeling** [Elbrechter et al., 2011a]

The paper was presented at the IEEE/ISJ International Conference on Intelligent Robots and Systems (IROS 2011) in San Francisco, California. An associated video can be found on the CITEC YouTube-channel [Elbrechter et al., 2011b].

[1] http://bulletphysics.org

4.1 Related Work

Work that is related to the particular aspects of picking up paper is presented here. Since the paper detection was performed using fiducial markers printed directly on the sheet of paper, different fiducial marker types and detection libraries were investigated. Another requirement was the modeling of paper deformation, and therefore common approaches for the modeling of deformable flat objects, often referred to as *thin shells* are presented. Once detection and modeling are discussed and anchored in their corresponding fields of research, an overview of alternative robot systems that are able to pick-up paper or deformable objects is given.

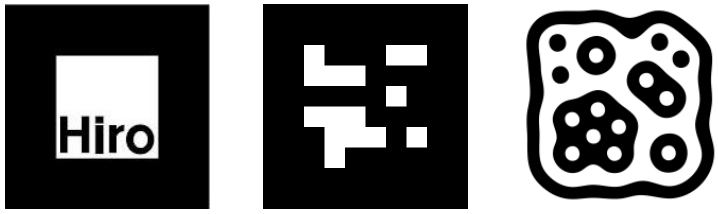
4.1.1 Perception using Fiducial Markers

In many computer-vision applications, invariably it arises that a processing step involves the detection and identification of key points in the visible scene so that they can be used as landmarks in further processing steps. This can prove to be very difficult for a number of reasons, such as having to deal with varying lighting conditions, partial object occlusions, unstructured and untextured image regions or even just due to poor image quality. To avoid these issues, the scene can be augmented using visual markers that can be detected more easily and more robustly. *Fiducial marker* detection is carried out in many varied image processing applications. A very general definition is:

Definition 1 (Fiducial Markers) *A fiducial marker (or short a "fiducial") is an object or an artificial cue that is added to a scene or to the used visual system so that it becomes visible in the produced images. Fiducial markers are usually used as reference points that are either manually or automatically well detectable.*

In times of analog photography so called *Reseau plates* [Brown, 1979] were commonly used for later image correction or image based measurements. Prominent examples of the usually cross-shaped fiducials on these are also visible on the photographs taken on the moon, that were sometimes misinterpreted as a potential proof that the image were faked [Windley]. Another very general example is the use in typography, where fiducial markers are used to align successively printed layers. In optical motion tracking systems, such as Vicon[2], reflective fiducial markers are used to achieve very high 3D position estimation accuracy [Windolf et al., 2008]. Fiducial markers are also commonly used in augmented reality (AR) [Zhou et al., 2008] and camera calibration scenarios [Atcheson et al., 2010]. Also in the field of medical imaging, in addition to vessels and bones that serve as *natural* fiducial markers, artificial fiducial markers are either attached to the human body externally or can be implanted. The markers provide reliable key points for different kinds of image-based treatments and diagnostics. In *imageing-guided radiation therapy*, fiducial markers, implanted into a tumor allow for more accurate results [Kothary et al., 2009]. Whenever the images of different imaging systems such as 3D ultra sound and magnetic resonance imaging (MRI) are to be aligned, natural and artificial fiducials are used as key-points for the alignment process [Porter et al., 2001].
Another common field is that of robotics. In bio-medics, surgery robots use fiducial marker to track certain parts of the human body during an operation [Howe and Matsuoka, 1999]. In many robotics research applications, fiducial markers are commonly used to bypass object detection, tracking and pose estimation issues [Bersch et al., 2011; Mutka et al., 2008; Steffen et al., 2010].

[2] Website: http://www.vicon.com



(a) (b) (c)

Figure 4.2: Different common fiducial marker designs. **(a)** An ARToolKit fiducial marker. IDs are encoded by the image inside the black rectangle **(b)** An ARTag/ARToolKitPlus fiducial marker. IDs are encoded by a BCH code represented by the central binary pattern. **(c)** An Amoeba fiducial marker. IDs are encoded by a characteristic image region containment structure.

The Principle of Fiducial Markers

The appearance of fiducials is preferably different to the rest of the scene, which facilitates the use of simple and computationally inexpensive heuristics for the detection. A simple example involves the use or a red circular dot. If other parts of the image do not share this appearance, simple pattern matching can be used to find the position of the red dot in the image. The higher the requirements become, the more complex the design of the fiducial markers must be. In order to avoid a situation in witch homogeneously colored regions are misinterpreted as a huge set of markers, a distinct border color can be defined [Cho and Neumann, 1998]. When using template matching for the detection, the border can simply be added to the template.
If more than one key point must be detected, the use of several red dots can lead to identification issues. This can be avoided by using several colors or several calibrated cameras. This, however, narrows the number of allowed colors in the actual scene and makes the detection more complex. Furthermore, using several colors does not scale very well because in common lighting conditions, only a small set of colors can be differentiated robustly[Schroder et al., 2012]. Cho and Neumann [1998] partly compensated these drawbacks by using concentric circles of different colors to increase the number of distinguishable IDs. However, this is only robust when lighting conditions are constant. These problems led us to define two main requirements that must be considered for the design of fiducial markers:

1. **Detectability** Fiducial markers must be robustly detectable in a computationally inexpensive manner. This can be split into two sub-disciplines: detection speed and robustness.

2. **Distinguishability** The fiducial marker design must provide a marker ID that allows several markers to be robustly distinguished.

Considering these requirements, the use of color information for image markers leads to several issues and is therefore not commonly used. In addition to the problem that only a small set of marker IDs can be encoded into the colors, changing lighting conditions and the presence of shadows lower the robustness of the detection significantly. While shadows can at least partly be compensated by using color spaces where the lightness of a pixel can be explicitly disregarded (such as the HLS color space), changing lighting conditions and in particular sun light make it almost impossible to define the color distribution of a marker. More advanced approaches, that use time-adapting color histograms partly compensate these issues, but cannot counteract sudden changes of the lighting conditions, such as the sun emerging from behind a dark cloud.

Monochromatic Fiducials

The issues with color-based fiducials led researchers to the idea of using structured monochromatic markers that are distinguished by their characteristic black and white appearance. Again, the appearance must be different from the remaining scene or at least it must be very unlikely to occur. Markers usually consist of two visual components, one that allows for easy detection and one that encodes marker IDs. A very early fiducial marker type was introduced with the open-source ARToolKit framework [Kato and Billinghurst, 1999], originally developed for augmented reality applications. The classical ARToolKit markers[3] (see Figure 4.2a) were distinguished by a thick bordered black rectangle on a white background. Inside the rectangle is an arbitrary gray-scale image surrounded by a white margin that is used to distinguish different markers. For marker detection, the ARToolKit searches for quadrangular black regions in a thresholded input image. For each detected marker, the inner marker region is rectified and compared pixel-wise to a set of registered reference images. Since the marker orientation is not known in advance, this step is performed for all four possible marker rotations. A global similarity threshold is used to reject non-marker regions. For each marker, a position and orientation in the 2D image plane is computed. Assuming known camera parameters, the warped shape of the marker boundary can also be used to compute the 6D marker pose in space with respect to the camera coordinate frame. In the background of augmented reality applications, the resulting homogeneous marker transform can be used to render objects in the marker frame as an image overlay, which gives the impression that the rendered objects are stuck to the marker. A dual camera mode is also available and can be used for real stereoscopic 3D visualization with head mounted displays. The ARToolKit library provides satisfying results for images with a limited number of large markers. However, in scenes with many small markers or cluttered backgrounds, both speed and detection accuracy is severely affected [Fiala, 2005]. High false positive rates as well as the low processing speeds arise from the rather simple method that is used to compare the rectified centers of potential markers with the set of registered markers. The more markers that are registered, the more image patches must to be compared. Furthermore, it becomes increasingly likely that the rectified image of a non-marker quadrangle coincidentally looks similar to one of the registered marker images.
Considering the drawbacks of the ARToolKit library, in 2005 Fiala [2005] introduced the ARTag library. Even though the appearance of the ARTag markers (see Figure 4.2b) is very similar to ARToolKit markers, the marker IDs are encoded in a much more sophisticated manner. Each marker center contains a 2D 6 by 6 pixel (36 bit) BCH binary code [Bose and Chaudhuri, 1960], which encodes a 10 bit marker ID. ARTag can also detect inverted (white border on a black background) markers, producing a total of 2048 possible markers. The use of BCH codes rather than simple image matching techniques results in several advantages in both detection speed and accuracy. For each potential marker found, the marker center is simply rectified into a 6 by 6 pixel binary image, whose pixel-vector then directly contains the marker's BCH code word (or, due to the unknown marker orientation, a $90°$, $180°$ or $270°$ rotated version). Since the processing time needed to decode a single 36 bit code word is constant, the marker detection speed no longer depends on the number of registered markers. If the homography based rectification step yields no more than a few errors in the resulting code word, the *self-error-correction* property of the BCH code enables the system to still obtain a valid marker ID. Furthermore, a valid 36 bit code is extremely unlikely to coincidentally occur in the scene. The ARTag library also came with several other optimizations such as using local and temporarily adapting thresholds to cope with varying or inhomogeneous lighting conditions. Although it was originally released as an open-source library it is no longer available.
The idea of using BCH code markers was also adopted by the ARToolKitPlus library, which was designed on top of the older ARToolKit [Wagner and Schmalstieg, 2007]. In contrast to ARTag,

[3] In March 2015, a new open-source version of ARToolKit, which also features other marker types, was released.

a different BCH code that encodes a 12 bit ID at the expense of a little less robustness was developed. In 2009 the authors came up with a new closed source library [Wagner and Schmalstieg, 2009] called *Studierstube Tracker*, which is mainly focused on the use on mobile devices.
A very different fiducial marker design was presented in the *libfidtrack*-library [Kaltenbrunner, 2009], who used the markers for tracking tangible objects in an experimental sound synthesis application. In contrast to the thus far discussed libraries, these markers (see Figure 4.2c) do not use different appearance aspects for detectability and ID encoding. Rather their markers are distinguished by a well defined region pattern that is to be found in an image region containment graph. All markers are bounded by a coarsely quadrangular but irregular black border on a white surface. The contained white image region defines the root region for the marker's region containment graph. The graph contains a set of black regions each containing 0 to 12 white sub-regions, which uniquely defines the marker's ID. The detection framework performs a connected component analysis of the binarized input image and creates the region-containment graph of the whole image. The markers are then detected and identified by an optimized graph matching implementation. For optimal detection properties, the design of the fiducial markers was optimized using an evolutionary algorithm [Bencina et al., 2005]. The topology based marker detection is very fast because the marker detection is not performed on the image data, but on the region-graph, whose data footprint is usually at least two orders of magnitude smaller than the input image. However, this also leads to a severe disadvantage of the markers. The topological image representation does not allow 2D positions of marker regions to be identified. Together with the irregular shape of the markers, this makes monocular 3D/6D marker pose estimation impossible. Furthermore, libfidtrack's GPL license hindered us from adapting their code for integration into ICL[4] and they also do not allow their markers to be used with other detection libraries.

Comparison and Benchmarks

To provide a better overview of what is available, we coarsely compared the different fiducial marker detection libraries (see Table 4.1). At a later stage in the development process, all underlying detection mechanisms were natively re-implemented in ICL. By these means, it was possible to not only provide a unified interface for all marker-types, but also to significantly reduce the implementation footprint. A benchmark that compares the performances of the original libraries with the ICL implementations was also conducted and the results are in the table.
Even though, from the list of existing marker detection libraries, the BCH-code based markers used by ARToolKit+ and ARTag performed best, we decided to develop a new marker layout with a new marker detection library. While ARTag could not be used as it was simply no longer available for download, ARToolKit+ was ruled out because of certain API-inconsistencies and the fact that it is neither supported by the authors nor its GPL-based license would allow the source-code to be integrated into ICL. In contrast to this, the results of the, at that time, newest ARToolKit version were simply not good enough. Libfidtrack was not used because of its infeasibility to handle monocular 3D/6D marker pose estimation and license incompatibilities. ARToolKit 5, which was released as open-source in March 2015, was not available at that point and is therefore not listed in the Table.
The new fiducial marker layout (see Section 4.2.1) developed for the picking up paper robotic system (see Section 4.5), is also listed in the comparison Table 4.1. At this early stage of the project, a re-implementation of a BCH-code-based marker detection library, similar to ARTag, seemed too time-consuming, so an approach based on region-containment-graphs, like that implemented in libfidtrack, was decided upon. In a later iteration of our detection framework a plugin for the detection of BCH-code-based markers was developed, integrated into ICL and used for the folding

[4] ICL uses the less restrictive LGPL-license

	ARToolKit 2.x	ARToolKit+	Studierst. Tracker	libfidtrack	New Markers[7]
Supported Platforms	all[1]	Win./Lin.	Win./Lin./Android	all	all
Distribution	bin./src	bin/src	bin.	bin/src	bin/src
License	GPL[2]	GPL	commercial	GPL[4]	LGPL
Detection Time (ICL) [3]	100ms [9] (23ms)	n./a.[8] (8ms)	n./a.[5]	8.7ms (9.5ms)	10ms
Marker count	**O**(10)[6]	2048	n./a.	206	60 ($\times$2)
Main issue for accuracy	marker count	marker size, image noise	n./a. n./a.	marker size, image noise	marker size, image noise
Main issue for speed	vis. & loaded marker count	large vis. markers, vis. marker count	vis. marker count n./a.	image noise	image noise

1.) Windows, OSX, Linux **2.)** a commercial Version called *ARToolworks* exists. In March 2015, this was released as ARToolKit 5 under the terms of LGPL **3.)** 1280 $\times$ 960 images with 20 visible markers (numbers in braces refer to the detection time of ICL's marker detection framework) **4.)** The *amoeba* markers cannot be used with other libraries **5.)** No free version available **6.)** There is not a theoretical limit, but detecting many markers at once leads to a very bad performance in both, speed and accuracy **7.)** Presented in Section 4.2.1 **8.)** Due to the no longer ongoing support, ArtoolKit+ could not be compiled on a recent Ubuntu Linux system anymore. The introduction paper provides some benchmark results, but they neither mention the full computer specs, nor the used image resolution. **9.)** Benchmark was coarsely estimated form the results presented in [Fiala, 2004]. Here different hardware specs and VGA image resolution was used)

Table 4.1: General comparison of alternative fiducial marker detection libraries. The detection times were measured on an Intel ®Xeon ®E5530 CPU, running at 2.4GHz, with 64bit Linux. The entry *main issue for accuracy* provides the main factor for decreasing the detection accuracy, i.e. false-positive, false-negative and wrongly estimated marker IDs. The entry *main issue for speed* gives the factors that influence the detection speed.

paper robotic system (see Section 5.2.1).

4.1.2 Modeling Paper

In the literature a number of different approaches for paper modeling can be found. Much research has focused on *origami modeling*, and due to its origins, this is particularly popular in Japan. Another field of research, in which paper and other thin deformable objects are modeled, is computer-graphics. In mathematics research, bent paper without creases can be well described by developable surfaces defined by a zero Gaussian curvature.

Origami Modeling

The origami-related publications extend from purely mathematical models that try to formalize aspects of origami-folding [Alperin, 2000; Hull, 2006], toolkits that facilitate origami construction [Lam, 2009; Shimanuki et al., 2009] or allow for an interactive manipulation of virtual origami models [Fastag, 2006; Ida et al., 2006; Tachi, 2009] to examples of applying origami-formalisms to real world applications [Cromvik and Eriksson, 2006; You and Kuribayashi, 2006]. Even though these publications provide an impressive set of mathematical formalisms, folding and manipulation axioms and even GUI-based origami editors, they do not strongly intersect with the work presented in this thesis. First of all, folding origami is only a very specific subset of possible manipulation scenarios with paper and usually strives for very complex target paper configurations. However, starting from an anthropomorphic robotics point of view, even the most simple manipulation primitives, such as picking up the sheet of paper or folding it in half are already extremely difficult to realize. Therefore, most origami formalisms are simply too complex to use as a modeling back-end for the targeted robotic manipulation system.

Vision-driven Modeling of Origami

Being able to automatically track the deformation of paper while it is iteratively folded could help to automatically derive folding instructions from demonstration. To this end, using visual input to update model parameters has already been addressed in origami-related research.
The work presented by Mitani [2006] looks astonishingly similar to the detection and modeling sections presented in this chapter. They use QR-codes printed on a standard sheet of paper to reconstruct the fold-configuration while the paper is iteratively folded. However, in contrast to the system presented here, they state no real-time constraints and they assume the absolute absence of occlusions. In their system, a single snapshot of the paper is taken and analyzed after each fold using a brute force search though all possible configuration spaces, which they admit becomes computationally extremely complex as more folds are iteratively applied. In contrast, the system presented here tracks the deformation of the paper at real-time frame-rates, which allows for the inference of *self-occluded* parts of the paper.
Kinoshita and Watanabe [2008] presented a system that was able to track an origami folding sequence based on silhouette detection. They assumed trivial color-based segmentation of the origami's front and back face and almost no occlusion of the paper by the human hand. By defining a set of restrictions to trackable folding operations, in order to avoid ambiguous intermediate states, the system was able to track several classes of possible fold types, such as *valley-folding* or *inside-reverse-folding*. The system did not run completely autonomously, i.e. some actions needed to be explicitly named by the folding person and the capturing of camera images had to be triggered manually. Due to its many restrictions, in particular the missing robustness against occlusions, it was deemed not very suitable for our requirements. Nonetheless, their system was used by [Miyazaki et al., 2010] to augment virtual origami models with realistic textures extracted from camera images.

Modeling Deformable Objects in Computer-Graphics

An important goal in the computer-graphics community is the realistic rendering of animations. To achieve this, physically plausible object trajectories and deformations are of paramount importance and necessitate the modeling of deformable objects. In particular, the physical modeling of cloth and hair has become very important in order to attain realistic rendering in computer-games and CGI-based visual effects in movies. Also, the modeling of deformable objects with high internal stiffness such as paper or rubber is very much in the focus of computer-graphics research. Deformable 2D surfaces are commonly modeled by *thin shell models* [Arnold, 2000; Ciarlet, 2000; Grinspun et al., 2003] and are characterized by curved objects with a negligible thickness. Existing systems allow for impressively realistic modeling, but are not yet applicable in real-time [Grinspun, 2005]. More recent approaches such as that proposed by Martin et al. [2010] allow for modeling several different aspects of deformation, such as bending, buckling, writhing, cutting and merging by using a unified simulation rule. However, they require very complex implementations and are only real-time-capable in very simple situations.
Since physically plausible modeling of paper is only one sub-aspect of this thesis, the choice was made to *use* and, where necessary, adapt the Bullet physics-engine [bul] as an existing simulation tool rather than to implement one from scratch.

Developable Surfaces

From a mathematical point of view, developable surfaces are well suited to describe deformed paper [Hilbert et al., 1952]. Developable surfaces in 3D space are always *ruled*, which means that

each point on the surface lies on a straight line that also lies completely on the surface. Assuming that paper is only bendable, but not stretchable, this applies perfectly to surfaces that can be created by bending or rolling a flat piece of paper. Huffman [1976] provides an additional definition based on Gaussian curvature: developable surfaces have a Gaussian curvature of zero. The Gaussian curvature of a 2D surface in 3D space at a point p is defined by the quotient $K = G/F$. F is defined by the area bounded by a small closed path around p, and G is the area of that path when it is mapped to the Gaussian sphere $\mathbf{S}^2$. Since a developable surface is locally always bent in one direction only, G and therefore also K is always zero. Huffman [1976] states that developable surfaces are well suited for modeling surfaces in computer aided design and manufacturing, because they are, compared to arbitrary surfaces, well defined and more powerful than planar surfaces. However, the missing intuitive link between a detected set of 2D/3D point correspondences and these definitions, led us to select a simpler mathematical model that represents the paper surface by smoothly interpolating between localized linear functions (see Section 4.3.2).

4.1.3 Robot Control

Picking up paper with anthropomorphic robot hands has so far not been addressed in research. Due to the complexity of the task, paper-like objects are usually picked-up and manipulated with very specialized tools or robots. In industry, paper, sheet metal, cardboard, cloth, food and other deformable objects are usually moved using rolls located on one or both sides of the surface. While thin metal plates can, dependent on the type of metal, also be grasped using magnetic grippers, vacuum grippers have to be used for paper. A common example of a machine that can pick up paper is a printer. The origami folding robot developed by Balkcom [2004] used air pressure to pick-up and move the manipulated sheet of paper. The dexterous paper folding robot developed by Tanaka et al. [2007] was not able to pick up paper. Instead, it used high and low friction fingertips to either fixate the paper at a certain position or to bend it.
In research towards automatic folding of cardboard-boxes, the picking-up problem is also often neglected by using special supply trays for the raw material [Dubey and Dai, 2006; Lu and Akella, 2000; Yao et al., 2011].
Other than this, picking-up deformable objects and materials has been mostly addressed from the perspective of manipulating laundry and cloth [Bersch et al., 2011; Maitin-Shepard et al., 2010; Van Den Berg et al., 2011]. It seems at first glance that this could be assumed to be very similar or at least easily transferable to the problem of picking up paper, but this turns out not to be the case. Cloth is usually thicker, more flexible and is only weakly plastic. This means that picking up a piece of cloth can be achieved by simply grasping it with a two jaw gripper, even if it lies flat on a surface. Applying a similar technique for picking up paper would add non-reversible creases to the paper with must be avoided.

4.2 Perception

In this section, a new detection and tracking framework for deformable surfaces is presented. While the detection and tracking engine for the paper-shifting experiment (see Section 2.2) assumed a rigid paper model linked to the 2D space of the table-top, a more complex tracking method is needed here. The new method not only generalizes to 3D, but also allows for modeling and tracking deformation of the sheet of paper. The tracking framework is inherently linked to the paper model that is used. In order to decouple tracking and visual detection, a generic interface was defined (see Figure 4.3). In each processing step, the vision module extracts a set of key-points $K = \{k_l\}$ each linking a point $\mathbf{k}_l^m \in \mathbb{R}^2$ on the model surface to an estimated point

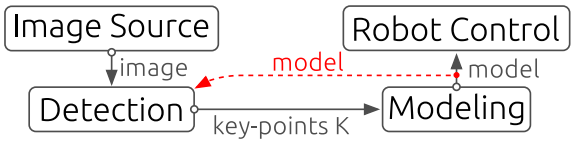


Figure 4.3: Interface between perception and modeling (based on Figure 2.4). The detection unit produces key-points $K = \{k_l\}$ that are sent to the modeling unit. The current model is not only used for the robot control, but also as feed-back for the detection module, which can exploit knowledge about the current model configuration to enhance detection speed and robustness by only searching for features in the model's vicinity.

$\mathbf{k}_l^w \in \mathbb{R}^3$ in 3D space. The set K is passed to the vision module. In turn, the modeling component provides feedback about the current belief (the current model), which can be used by the visual detection engine to narrow the search space for features if required. This feedback channel is used by the RGBD-point-cloud based detection engine described in Section 6.2.

4.2.1 Marker-based Detection of a Deformed Sheet of Paper

The task of the vision module is to find the set of key-points $K = \{(\mathbf{k}_l^m, \mathbf{k}_l^w)\}$, each linking a 2D position on the paper to a 3D position in the world. 3D information can be obtained by different means. Monocular vision systems can provide 3D information, however usually with a poor accuracy along the camera's view-axis. In contrast, calibrated multi-camera systems can be used to much more accurately estimate 3D positions using triangulation techniques. However this assumes known point correspondences in the different camera images, which can be difficult to compute. By using fiducial markers, printed directly on the tracked sheet of paper, several issues are significantly simplified all at once. First, the fiducial markers can be efficiently detected in the input images. Second, each marker encodes an ID, which can be used to solve the correspondence problem for the multi-camera setups and lastly, by memorizing the marker layout printed on both sides of the paper, the correspondence problem between 3D world positions and 2D model positions is also explicitly solved.
Another obvious source for acquiring accurate real-time 3D data are 3D cameras such as the Microsoft Kinect camera or Mesa Imaging's time-of-flight camera Swiss-Ranger[5]. However, while these devices provide accurate and dense 3D data at low computational cost, it remains unclear how to estimate the corresponding 2D paper coordinates for the 3D points. Furthermore, it is worth mentioning that the decision to use markers was made before the existence of an open-source interface to acquire 3D point cloud data from Kinect. For marker-less tracking, based on RGBD point cloud data, an additional tracking framework was developed and evaluated (see Section 6.2). As discussed before, when the existing fiducial marker detection frameworks were looked at, it turned out that none of them were well suited to the desired task (see Section 4.1.1). Therefore, a new marker layout was created that allowed for fast detection with a low false positive rate and features a built-in mechanism for 6D pose estimation.

Region-Containment based Marker Detection

The new fiducial marker design is very similar to the *Amoeba*-markers introduced by Kaltenbrunner [2009]. The markers are detected by their unique image region containment structure derived from a topological region containment graph that is extracted from a binarized input image. The processing pipeline is sketched in Figure 4.4.

[5] http://www.mesa-imaging.ch

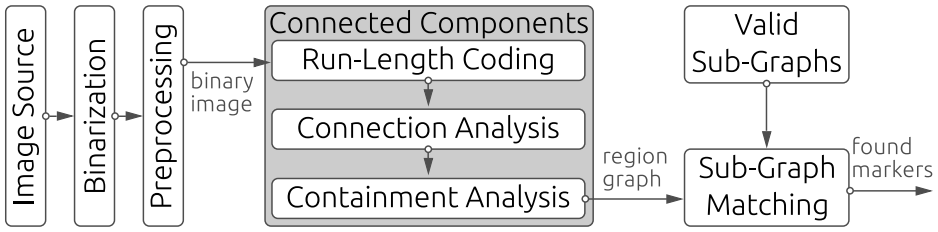


Figure 4.4: Tool-chain for the fast fiducial marker detection. The binarized and de-noised input image is fed into the connected component module. Here, the region connectivity is estimated based on a preceding run-length coding phase. With the resulting connectivity information, a containment graph is computed that allows markers to be found by means of sub-graph matching.

The gray-scale input image is first binarized using a locally adaptive threshold operator, which compares each pixel to the average gray-value in its neighborhood. The average values can be efficiently computed from the gray image's integral image [Viola and Jones, 2001]. The adaptive threshold leads to much better binarization results in case of shadowy image regions than a standard global threshold operation. The use of adaptive threshold techniques was already suggested and evaluated by [Fiala, 2005]. Subsequently a binary median operator is applied to the binary image to remove very small regions. The connected component analysis first transforms the binary image into a run-length-encoded (RLE) representation. Here, each image line is represented by a list of *runs*, each defined by a gray-value, an offset, a length and the line's y-index. This can be implemented very efficiently and allows for the reduction of the memory footprint of the binary image to a fraction of its original size. By conservatively assuming an average run-length of 100 pixels, a 1KB-line of a 1000×1000 image can be represented by only 80 bytes[6]. RLE-based connected component analysis was shown to outperform traditional methods by a factor of ten [He et al., 2008]. The estimation of the region-connectivity can then efficiently be implemented on the basis of the RLE-representation. More details can be found in the ICL API documentation[7]. The connectivity analysis results in a list of *regions* each defined by a set of corresponding RLE-runs (necessary to compute region features, such as pixel-count or center of gravity) and a set of adjacent regions, including special regions that represents the image borders. Region containment is defined as follows:

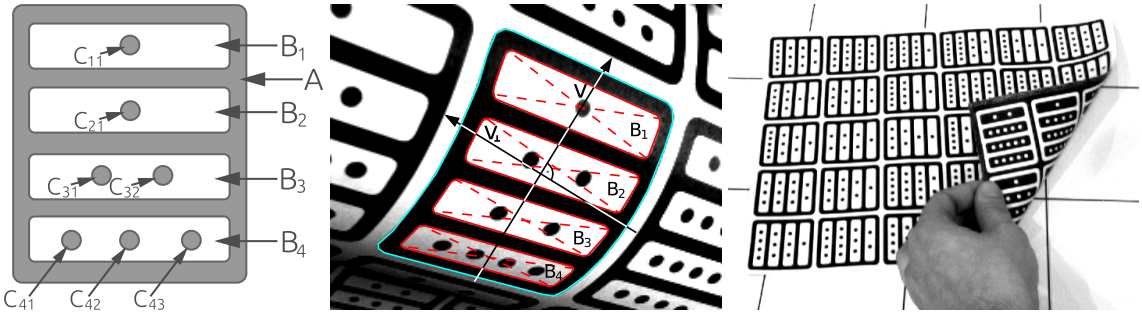
Definition 2 (Region Containment) *An image region A contains an image region B if there is no path along the image's connectivity graph that allows it to reach the image border when starting from B without passing A. If A and B are adjacent, A contains B* ***directly*** *otherwise, A contains B* ***indirectly****.*

Due to the transition from $\mathcal{O}(10^6)$ image pixels to a maximum of $\mathcal{O}(10^3)$ image regions, it is possible to pre-compute the region-containment graph, by endowing each region structure with a pointer to its parent region and a list of directly contained child regions.

New Fiducial Marker Layout

The geometrical structure of the new markers is designed to allow for an easy identification of each single marker region by means of simple geometric heuristics. The topology of each marker is defined by a top-level region, A (see Figure 4.5a) that includes four child-regions, B_i. In turn, each B_i contains an additional set of one to five 3rd-level regions C_{i1} to C_{iN_i}, i.e. B_i contains

[6] If the four members of each run are represented by 16bit values (i.e. two bytes per value)
[7] www.iclcv.org



(a) (b) (c)

Figure 4.5: New fiducial marker design, region layout and identification **(a)** Each marker ID is defined by its unique structure in the region containment graph. The top-level region A contains four sub-regions B_1-B_4 each containing one to five 3rd-level regions C_{ij}. **(b)** The vector v is used to sort the B_is. Then $v_\perp$ is created perpendicular to v in order to sort the C_{ij}. **(c)** Used sheet of paper. Back-face markers are inverted, which allows the use of identical and aligned markers on both sides of the sheet of paper.

exactly N_i 3rd-level regions. A marker's ID is encoded by the combination of the numbers N_i of 3rd-level regions. This allows for defining the marker ID as a four-tuple $M = (N_1, N_2, N_3, N_4)$. As an example, the marker shown in Figure 4.5a has the ID $N = (1, 1, 2, 3)$. Each N_i has five possible values leading to a total for $4^5 = 1024$ distinguishable marker IDs.

Selection of used Marker IDs

In order to avoid mixing up markers and to increase the detection accuracy, additional heuristics for the choice of actually used IDs had to be introduced. Due to the fact that the sheet of paper (and therefore also the makers) can be arbitrarily rotated with respect to the camera, IDs that are ambiguous with respect to a reversion of their code M could not be used. This is achieved by only allowing markers that have a monotonic increasing number of 3rd level regions: $N_i \leq N_{i+1}$ and where not all N_i are equal: $N_1 < N_4$. Thus, the markers get a direction, well defined by a *light* top part that contains less 3rd-level regions as the *heavier* bottom part. This restriction reduces the number of possible IDs to 120.
In order to increase the detection accuracy by decreasing the false-positive detection ratio, an error detection mechanism was implicitly added by using only a subset of the remaining 120 marker IDs. In the marker detection step, the 3rd level regions are smallest and therefore most difficult to detect. On the one hand, image noise – in particular existing in shadowy image regions – can be strong enough to show up as phantom regions even after median-based noise reduction. On the other hand, 3rd-level regions that are tilted away from the camera plane or too far away can easily be eliminated in the noise reduction layer. In both cases, this results in a wrong marker ID. In order to allow for the detection of such recognition errors the set of used markers was further reduced. The final set of used marker IDs contains only 60 markers, whose pairwise hamming distance $d(M^1, M^2) = \sum_i |N_i^1 - N_i^2|$ is greater than one. This means that wrongly detecting a single 3rd-level regions by either missing an existing one or detecting a region that does not actually exist, implicitly leads to an unused marker ID.

Paper Layout

For the robotic experiments conducted, a printed A4 sheet of paper with 5×6 markers on both sides was used. Even though this exactly equals the number of possible markers, only 30 different IDs were actually used. Rather then using the available 60 markers to cover both sides of the paper, markers with an inverted color scheme (i.e. using white A and C_{ij} regions and black B_i regions) were used on the paper's back-face (see Figure 4.5c), making it possible to perfectly align every single front and back face region. This also allowed for joining marker detection results of cameras that *see* different sides of the sheet of paper. The order of used markers was randomized to have a more homogeneous density of C_{ij} on the paper.
The 2D-paper coordinate frame was defined on the front face with its origin in the top left corner. If the paper is seen in portrait format, the positive x-axis points to the right and the positive y-axis points downwards. The paper size in 3D space, is given by $w \times h = 0.210m \times 0.297m$.

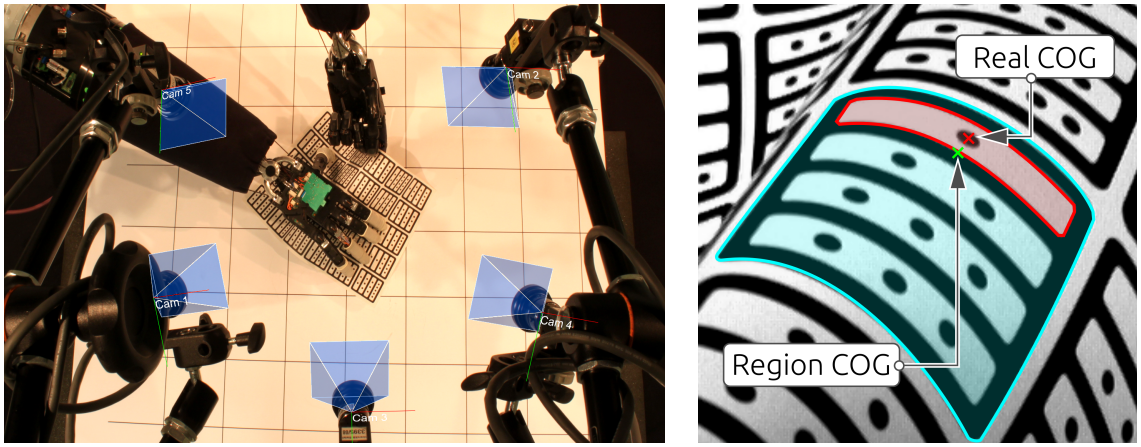
Sub-Graph-Matching in the Region-Containment-Graph and Region Identification

In order to combine the detection results of different cameras to obtain 3D marker positions, the regions must be identified. Then corresponding regions, belonging to markers that were detected in at least two camera views, can be joined for triangulation to get 3D key-points. The known layout, which defines exactly where markers were printed on the sheet of paper, then provides information about the corresponding 2D model coordinate.
In the sub-graph matching step, only regions with four sub-regions are used as potential top-level regions A. As a first criterion, potential A regions are filtered out, if one of their sub-regions, B_i, contains less than one or more than five child-regions. In addition, some hand-tuned heuristics are applied to filter out too small, too large and too elongated regions. Then, the counts of sub-sub-regions, C_{ij}, are sorted in ascending order to estimate the marker ID, $M = (N_1, N_2, N_3, N_4)$. If M is a valid marker ID (i.e., part of the set of the 30 markers actually used), A and its corresponding ID is stored in the list of found markers. This step does not use geometrical, but only topographical information. In the next processing step, all marker regions B_i and C_{ij} are identified by means of geometric heuristics. First, the centers of gravity $\mathbf{b}_i$ and $\mathbf{c}_{ij}$ of the regions B_i and C_{ij} are estimated[8]. Once all $\mathbf{b}_i$ are known, the vector $\mathbf{v}$, connecting the centers $\mathbf{b}_m$ and $\mathbf{b}_n$ with largest pair-wise distance, is computed (see Figure 4.5b):

$$\mathbf{v} = \mathbf{b}_m - \mathbf{b}_n, \quad \text{with} \quad (m, n) = \arg\max_{(k,l)} \|\mathbf{b}_k - \mathbf{b}_l\|$$

The direction of $\mathbf{v}$ is defined by always letting it point towards the $\mathbf{b}_i$ where the corresponding region B_i has fewer child regions C_{ij}. The B_is are then sorted along $\mathbf{v}$ according to their center of gravity's inner product $\mathbf{b}_i^\tau \mathbf{v}$. Subsequently, for each B_i, the set of child-regions C_{ij} is sorted along $\mathbf{v}_\perp$ (perpendicular to $\mathbf{v}$) according to the inner products $\mathbf{c}_{ij}^\tau \mathbf{v}_\perp$ (see Figure 4.5b). For back-face markers (identified by a white top-level region A) $-\mathbf{v}_\perp$ is used in order to compensate the mirror effect caused by the two-sided print. By these means, front and back side regions are automatically ordered in the same direction, thus regions detected on opposite sides of the paper are implicitly associated correctly.

[8] Note that the underlying RLE-representation of the region pixels allows this step to be significantly sped up, because the center of gravity of all pixels is identical to the weighted center of gravity of all of the RLE code runs



(a) (b)

Figure 4.6: **(a)** Calibrated camera setup for robust 3D key-point detection. The use of five cameras leads to a better coverage of the whole scene and partly compensates occlusions. The used PointGrey Flea2G cameras use the IEEE-1394-B (FireWire-800) interface to provide Quad-VGA (1280×960) images at 30Hz. **(b)** Offset between real center of gravity (COG) of a bent region and the COG of the projected image region. The more a region is bent, the less accurate its COG can be estimated in the projected image.

4.2.2 3D Key-Point Estimation

For 3D estimation of key-points, a calibrated multi-camera setup is used (see Figure 4.6a). Even though two cameras would in general be sufficient for triangulation-based 3D estimation, five cameras are used. This leads to a much better coverage of the scene and significantly reduces the area of the paper that gets completely occluded by the robot hands.
The marker detection algorithm is applied to each camera image. For 3D estimation, only the smallest regions, C_{ij}, of the markers are used. Due to possible deformation of the paper the centers of gravity of larger regions cannot be accurately estimated (see Figure 4.6b). Therefore, given that even small errors in the pixel domain can result in large errors in the 3D estimation, the centers of gravity of the larger regions, A and B_i, are disregarded. For each marker, three cases are distinguished:

1. The marker was not detected in any of the camera images
2. The marker was detected in exactly one camera image
3. The marker was detected in at least two camera images

While markers that are not detected are simply disregarded, two different 3D estimation methods are used in the cases 2 and 3. If a marker is only detected by a single camera, all its regions are processed conjointly. By assuming the marker to be planar[9], *planar pose detection from single view* methods can be used. If a marker is detected by more than one camera, the 3D position of each of its regions is estimated separately by joining the multi-view information.

[9] This seems to be conflictive to the observation of falsified centers of gravity in case of bent markers, but tests revealed that the assumption leads to acceptable errors.

Planar 3D Estimation from a Single View

3D pose estimation for a known planar target from a single view is commonly performed in two steps [Kato and Billinghurst, 1999; Schweighofer and Pinz, 2006; Wagner and Schmalstieg, 2007]. First, an initial *guess* is computed that usually minimizes an algebraic error. In order to minimize the projection error, i.e., the MSE between detected and projected key-points, heuristics and non-linear optimization techniques are applied to refine the initial result.
In our system, the closed-form-solution suggested by Yang et al. [2009] is used to obtain an initial pose. Here, the problem is transferred into a 2D homography estimation problem that needs at least four points. The method uses the camera matrix, the image pixel coordinates, $\mathbf{c}_{ij}$, of the region centers C_{ij} and their corresponding local 2D coordinates, $\mathbf{l}_{ij}$, on the paper/marker surface. The latter are derived from the known layout of the printed paper by mapping the centers of the printed regions, C_{ij}, to a local 2D coordinate frame positioned at the marker center. If a region C_{ij} is centered at $\mathbf{c}_{ij}^m$ on the paper surface, and the corresponding top-level region A is centered at $\mathbf{a}^m$, the local coordinates are obtained by a simple shifting operation

$$\mathbf{l}_{ij} = \mathbf{c}_{ij}^m - \mathbf{a}^m.$$

The method is fast and it provides very good results given optimal input data, but it is very sensitive to errors in the input data. Therefore a non-linear optimization of the resulting pose is applied, using the result of the closed-form solution as initialization. Due to it's simplicity a simplex optimization based method [Press et al., 1992] was used here.
Once the homogeneous transformation matrix T of a marker is known, it can be used to compute the 3D coordinates of the maker regions. To this end, the regions' 2D local planar coordinates $\mathbf{l}_{ij} = (x_{ij}, y_{ij})$ are assumed to be 3D with $z = 0$, leading to homogeneous vectors $\mathbf{l}'_{ij} = (x_{ij}, y_{ij}, 0, 1)$. The desired 3D world coordinates of the regions can then be computed by

$$\mathbf{c}_{ij}^w = T\mathbf{l}'_{ij}.$$

By associating the original model coordinates $\mathbf{c}_{ij}^m$ with the computed 3D world coordinates $\mathbf{c}_{ij}^w$, each marker detected in exactly one camera image allows for the computation of a set of key points

$$\{(\mathbf{c}_{ij}^m, \mathbf{c}_{ij}^w) | i \in \{1, .., 4\} \text{ and } j \in \{1, .., N_i\} \}.$$

Multi-View Position Estimation

The 3D positions of regions that belong to markers detected in at least two cameras are estimated using a *direct linear transform* based triangulation method. Given $n > 1$ camera matrices

$$Q_i = [R_i|\mathbf{t}_i] = \left[\begin{array}{c|c} \mathbf{x_i} & t_i^x \\ \mathbf{y_i} & t_i^y \\ \mathbf{z_i} & t_i^z \end{array}\right],$$

where $\mathbf{x_i}$, $\mathbf{y_i}$ and $\mathbf{z_i} \in \mathbb{R}^3$, the projection of a point $\mathbf{p}^w$ in the image of camera i is given by

$$(u_i, v_i) = \left(\frac{\mathbf{x}_i\mathbf{p}^w + t_i^x}{\mathbf{z}_i\mathbf{p}^w + t_i^z}, \frac{\mathbf{y}_i\mathbf{p}^w + t_i^y}{\mathbf{z}_i\mathbf{p}^w + t_i^z}\right).$$

For a given set of camera matrices $\{Q_i\}$ and corresponding image projections $\{(u_i, v_i)\}$, each entry provides two constraints for the estimation of the unknown 3D position $\hat{\mathbf{p}}^w$. By reformulating the problem in matrix notation $A\hat{\mathbf{p}}^w = B$, with

$$A = \begin{bmatrix} u_1\mathbf{z}_1 - \mathbf{x}_1 \\ v_1\mathbf{z}_1 - \mathbf{y}_1 \\ u_2\mathbf{z}_2 - \mathbf{x}_2 \\ \cdots \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} t_1^x - u_1 t_1^z \\ t_1^y - v_1 t_1^z \\ t_2^x - u_2 t_2^z \\ \cdots \end{bmatrix},$$

where A is a $2n \times 3$ matrix and B is a $2n$-dimensional column vector, the problem can be solved by $\hat{\mathbf{p}}^w = A^+B$, where A^+ is the pseudo inverse of A.
In the present marker region detection scenario, the image coordinates (u, v) are given by the estimated centers of gravity $\mathbf{c}_{ij}$ of the regions C_{ij}.

4.3 Modeling

The schematic depicted in Figure 4.3 is now used as an interface to implement two different modeling techniques. The first model that was implemented is a purely mathematical model, which is explicitly defined to be as simple as possible. The model represents the 2D paper manifold by a finite set of 2D polynomial functions embedded into 3D space. The surface is smoothed using soft-max interpolation. Furthermore, the model's resulting surface function, p, is at a given time-step directly defined by the current set of key-points provided by the perception module. The mathematical model is compared to a more sophisticated physical model that was implemented using the *soft-body-physics* module of the Bullet physics engine[10]. In contrast to the mathematical model the physics model uses a large set of constraints that explicitly limit local curvature and provide distance preservation and even temporal tracking. Therefore, the physical model's surface function p is only indirectly connected to the set of key-points provided by the perception module. For the connection between the real-world observation and the model, a special control law is defined.

4.3.1 Prerequisites

Before the vision-system or the modeling engine can be introduced properly, an abstract paper model has to be defined. By disregarding the paper's thickness, it can be modeled by a 2D manifold, embedded into 3D space (see Figure 4.7). The paper model is assumed to be bounded and to have a rectangular shape, leading to the a parametric paper function

$$p : P \to \mathbb{R}^3, \quad \text{where} \quad P = [0, w] \times [0, h]. \tag{4.1}$$

The paper function p transforms 2D-model coordinates (denoted by $\mathbf{x}^m$) into 3D world coordinates (denoted by $\mathbf{x}^w$). A further requirement is that the latent parameters of p are compatible to the metric of the embedding 3D space, i.e. moving small distances along the paper surface always results in a comparable movement, in terms of the distance, in the 3D world space

$$\lim_{\Delta\mathbf{x}^m \to \mathbf{0}} \|p(\mathbf{x}^m) - p(\mathbf{x}^m + \Delta\mathbf{x}^m)\|_3 \approx \|\Delta\mathbf{x}^m\|_2. \tag{4.2}$$

This constraint will be referenced as *distance preservation*. The rule calls for a *comparable* rather than for an *equal* distance to take into account that paper can be stretched minimally. Furthermore, the surface is required to be smooth, i.e. it has a limited local Gaussian curvature

$$K(\mathbf{x}^m) < K_{\text{max}} \in \mathbb{R} \quad \forall \mathbf{x}^m \in P. \tag{4.3}$$

$K(\mathbf{x}^m)$ can be derived from the Gauss map $g : P \to \mathbb{S}^2$ that maps from a given model surface position to a normalized surface-normal vector on the unit sphere $\mathbb{S}^2$ [Sullivan, 2005]. The constraint would be violated by *hard* folds. However, it can be argued that a fold can always be approximated with an arbitrary desired accuracy by a *large-enough* finite curvature. The model constraints would also have to be adapted if also cutting or ripping the paper was to be modeled.

[10] http://bulletphysics.org

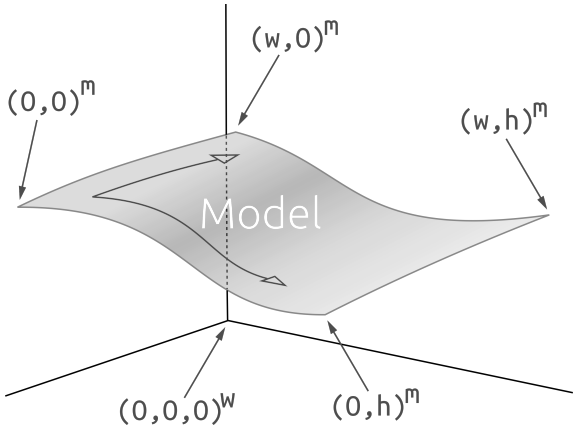


Figure 4.7: Rectangular 2D paper manifold embedded into 3D space.

In contrast to *developable surfaces* [Huffman, 1976] that are defined by a zero Gaussian curvature, only a limited Gaussian curvature is claimed here. In addition to the fact that real paper is minimally stretchable [Balkcom, 2004], which violates the zero Gaussian curvature constraint, the additional constraint would make the modeling process much more complicated.
If the modeling affords (or implicitly leads to) a different parametrization with parameter space P', an invertible mapping $\mu : P' \rightarrow P$ must be provided so that equation 4.2 becomes true for $p'(\mathbf{x}^m) = p(\mu^{-1}(\mathbf{x}^m))$.

4.3.2 The Simple Geometric Mathematical Model

The soft-max interpolated polynomial model approximates the paper surface by a set of 2D polynomials

$$\mathbf{P}_i : \mathbb{R}^2 \rightarrow \mathbb{R}^3,$$

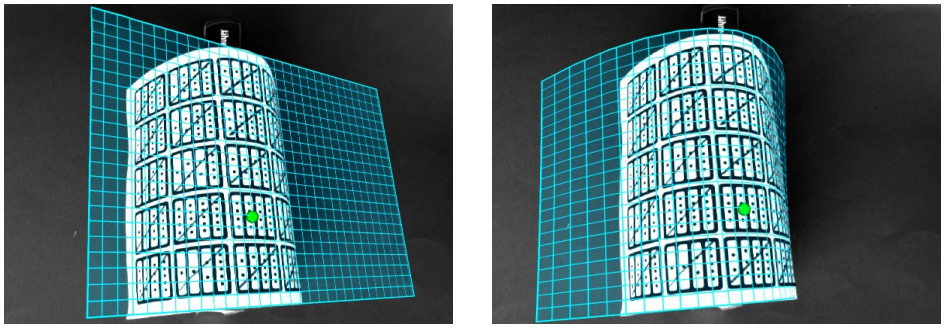
embedded into 3D space. Each $\mathbf{P}_i$ maps 2D paper model coordinate $\mathbf{x}^m$ to a corresponding real-world position $\mathbf{x}^w$, but only approximates well for a small patch of the paper surface. For each detected marker M_i, a polynomial, $\mathbf{P}_i$, is fitted into the set of key-points $\{(\mathbf{k}_l^m, \mathbf{k}_l^w) | l \in \{1, .., n_i\}\}$ (see Section 4.2.1) computed from the regions, associated with marker M_i. These keypoints formalize the output of the vision system, which estimates 3D world coordinates $\mathbf{k}_1^w..\mathbf{k}_{n_i}^w$ of the detected 2D positions of the regions of each Marker M_i.
The polynomials $\mathbf{P}_i$ are defined as a weighted superposition of scalar basis functions $b_j(\mathbf{x}^m)$

$$\mathbf{P}_i(\mathbf{x}^m) = A_i(b_1(\mathbf{x}^m), b_2(\mathbf{x}^m), \ldots, b_p(\mathbf{x}^m))^\tau. \tag{4.4}$$

The $3 \times p$ parameter matrix A_i can be estimated by standard regression to solve $W = A_i M$, with

$$W = (\mathbf{k}_1^w, \mathbf{k}_2^w, \ldots, \mathbf{k}_{n_i}^w) \quad \text{and} \quad M = \begin{bmatrix} b_1(\mathbf{k}_1^m) & b_1(\mathbf{k}_2^m) & \ldots & b_1(\mathbf{k}_{n_i}^m) \\ b_2(\mathbf{k}_1^m) & b_2(\mathbf{k}_2^m) & \ldots & b_2(\mathbf{k}_{n_i}^m) \\ \ldots & \ldots & \ldots & \ldots \\ b_p(\mathbf{k}_1^m) & b_p(\mathbf{k}_2^m) & \ldots & b_p(\mathbf{k}_{n_i}^m) \end{bmatrix}.$$

Here, W is a $3 \times n_i$ matrix and M is a $p \times n_i$ matrix. Using the pseudo-inverse M^+ of M leads to a least-square solution $A_i = WM^+$.



(a) (b)

Figure 4.8: Comparison of the approximation of local marker patches by polynomials of different complexity. Only a single marker (marked by the green sphere) is detected here; other markers were deactivated by drawing a line through them **(a)** Polynomial with three basis functions. The marker surface is approximated by a plane, which leads to large errors at model positions further away from the detected marker if the paper is curved **(b)** More complex polynomial using six basis functions. Here, the marker surface is approximated by a quadratic function allowing for a better approximation of the paper surface, even for model coordinates further away from the detected marker.

Choice of Scalar Basis Functions

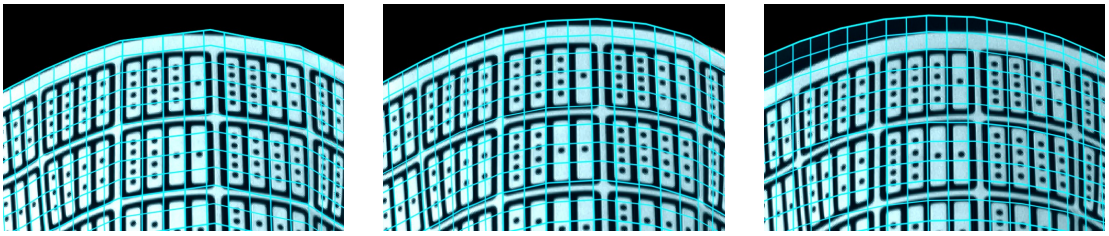
The mathematical approximation of model patches by polynomials described by Equation 4.4 allows us to implicitly change the model complexity by altering the set of used basis functions b_j. Let $\mathbf{x}^m = (x_1, x_2)^\tau$, the minimal set of three basis functions

$$\begin{aligned} b_1(\mathbf{x}^m) &= 1 \\ b_2(\mathbf{x}^m) &= x_1 \\ b_3(\mathbf{x}^m) &= x_2 \end{aligned}$$

models each marker patch by a plane. While adding more polynomial terms increases the flexibility of the $\mathbf{P}_i$, the use of less basis functions leads to unusable results. In particular, if only b_1 is used, each marker patch is approximated by a single point in 3D space, making it impossible to extrapolate model edges correctly, because these are not located *between* detected key-points. Therefore, in addition to the b_js defined for the linear function $\mathbf{P}_i$, an extended set was evaluated, which adds three extra basis functions

$$\begin{aligned} b_4(\mathbf{x}^m) &= x_1 x_2 \\ b_5(\mathbf{x}^m) &= x_1^2 \\ b_6(\mathbf{x}^m) &= x_2^2. \end{aligned}$$

Here, each $\mathbf{P}_i$ can model curvature and, due to the mixed term $x_1 x_2$, even twists. However, adding more degrees of freedom not only leads to a better local approximation (see Figure 4.8), it also makes the model more prone to overfitting, which is the reason why polynomials with cubic basis functions were not used.



(a) (b) (c)

Figure 4.9: Influence of the soft-max interpolation bandwidth parameter σ **(a)** A small value of σ leads to *winner takes all* interpolation, resulting in a piece-wise linear approximation of the paper surface. **(b)** A larger value of σ interpolates between the linear pieces $\mathbf{P}_i$ and therefore leads to a smooth surface. **(c)** If σ becomes too large, the model becomes *stiffer*, which limits the maximum curvature that can be modeled

Soft-Max Interpolation

Once each detected marker surface is approximated by a polynomial function, $\mathbf{P}_i$, the paper surface function, p, is defined using soft-max interpolation. For this, a Gaussian kernel

$$\mathrm{g}_i(\mathbf{x}^m, \sigma) = \exp(-\frac{\|\mathbf{x}^m - \mu_i^m\|^2}{2\sigma^2})$$

is centered at each center μ_i^m of a detected marker, M_i. The paper surface function is defined by

$$p(\mathbf{x}^m) = \frac{\sum_i \mathrm{g}_i(\mathbf{x}^m, \sigma) P_i(\mathbf{x}^m)}{\sum_i \mathrm{g}_i(\mathbf{x}^m, \sigma)}. \tag{4.5}$$

The single parameter σ can be used to adjust the smoothness of the model (see Figure 4.9). While a very small σ leads to a piece-wise linear paper surface function, values in the order of the real size of a paper surface patch covered by a marker, provide good interpolation results. If σ is too high, all linear functions get a similar contribution at each point on the surface, leading to more or less linear paper function p. Therefore, σ indirectly defines the *stiffness* of the paper model.

Model Properties

The maximum local curvature of the mathematical model is implicitly defined by the soft-max interpolation bandwidth σ. Setting σ to zero would not only violate the local curvature constraint (see Equation 4.3), but also lead to numerical issues. The choice of polynomial basis functions also indirectly affects the curvature of the resulting paper function.
The distance preservation rule (see Equation 4.2) is not handled optimally by the mathematical model. While the polynomials strictly include the distance preservation property in their minimized errors, the soft-max interpolation mechanism does not (see evaluation in Section 4.4.4). In some special cases, this leads to very implausible modeling results. More details and some qualitative results are given in Section 4.4.

4.3.3 Physics-Based Modeling

After evaluating the mathematical model (See Section 4.4.2 et seq.), it became clear that extra constraints were needed to make the modeling more plausible and indeed more stable. The main

issues of the mathematical model are:

- Missing mechanism to ensure distance preservation
- No temporal tracking
- Extremely bad approximation of the parts of the paper that are not visible
- Inflexible global interpolation bandwidth parameter
- Difficult to extend

By adding more and more explicit constraints to the mathematical model that allow for compensating one or several of these issues, one would basically approach an approximation of the actual physics of the paper. So rather than trying to extend the mathematical model, an alternative model was implemented using the Bullet physics engine[11]. This allows for optimizing the model behavior from a logical perspective of a physical object. The interface of the physics engine was taken as a basic layer to avoid having to deal with the implementation of the physical behavior of the paper and the interaction with possible other objects manually. The internal implementation of the physics engine was not analyzed.

The Bullet Physics Engine

Bullet is an open source physics engine provided under the terms of the zlib license and was mainly written by Erwin Coumans. In addition to standard rigid-body collision and dynamics, it provides a soft-body physics module. Bullet has been used in game engines, physics and graphics demos and even for rendering special effects in Hollywood movies. Many 3D modeling and animation tools use Bullet plugins for physics simulation.
Bullet was preferred to other physics engines because of its easy-to-use soft-body physics module, which was needed for the paper simulation. But also the free license and the active community accessible via an associated internet forum contributed to the decision to use it. In addition, Bullet is known to be fast and accurate. The version used for the work in this thesis was 2.8. A new major version, featuring a GPGPU-based rigid body physics pipeline, has already been featured at the recent Game Developers Conference (GDC 2013).

A Soft Body Physics Model of Paper

In the Bullet engine soft-body objects are defined by a set of nodes, a set of parametrized links and a set of faces. Each link connects two nodes and is set up with a resting distance and a stiffness parameter. Links that connect adjacent nodes provide distance preservation, by defining a desired distance between the two referenced nodes. Links that span over several nodes provide a bending stiffness. Both effects can be adjusted by altering the link's stiffness parameter. Faces, connecting either 3 or 4 nodes, are defined by a resting area. For physical simulation, each node is defined by its mass, its position, its velocity and its current acceleration. Each dynamic factor such as reacting to external forces, using links or using faces can be separately enabled.
The implemented paper model is a 2D-specialization of the general soft-body physics object in Bullet. The model is defined by a regular 2D grid of nodes $n_{\mathbf{p}^m}$, $\mathbf{p}^m \in P' = \{1,..,W\} \times \{1,..,H\}$. According to the abstract model defined in Section 4.3.1, $\mathbf{p}^m$ defines the model space coordinate of a node. The resulting paper function $p : P' \rightarrow \mathbb{R}^3$ is defined by the piecewise bi-linear interpolation of the model grid. The mapping $\mu : P' \rightarrow P$ that maps from the natural numbered model space P' to the metric space P, can easily be derived from the grid size $W \times H$ and the actual size of the modeled sheet of paper. The position and the velocity of node $n_{\mathbf{p}^m}$ will

[11] http://bulletphysics.org

be denoted by $\mathbf{x}^w_{\mathbf{p}^m}$ and $\mathbf{v}^w_{\mathbf{p}^m}$ respectively. In order to allow for a direct mapping between detected marker positions and physical model nodes, each marker patch is modeled by 2×2 model cells[12].

Moving The Model According to the Observation

Once the model is defined, a tracking mechanism is needed that moves the model according the current detection results, i.e., the set of key-points $K = \{(\mathbf{k}^m_l, \mathbf{k}^w_l,)\}$, which associate the model and world space positions at the current time step. In order to move the model nodes that correspond to detected marker centers[13] there are three physical options:

1. Move nodes by controlling the physical node's positions
2. Move nodes by controlling the physical node's velocities
3. Move nodes by controlling the physical node's accelerations

Even though force-based control, i.e. accelerating nodes towards their target positions seems like the obvious choice, a velocity based control was used. The main argument against using the acceleration-based control approach is that the existing node inertia would have to be compensated by an overdrive mechanism. This effect was verified in test experiments, which showed that acceleration-based control is too unstable. Controlling node positions directly also leads to extremely unstable model behavior.
For the velocity-based model control law, it turned out that a simple P-controller,

$$\mathbf{v}_{\mathbf{k}^m_l} = \lambda(\mathbf{k}^w_l - \mathbf{x}^w_{\mathbf{k}^m_l}), \tag{4.6}$$

parametrized by a single scalar proportional gain, λ, provided very good results. For each used key-point in the model space coordinates, $\mathbf{k}^m_l$, the velocity, $\mathbf{v}_{\mathbf{k}^m_l}$, of the corresponding model node is set to the distance vector (weighted by λ) between the current node position, $\mathbf{x}^w_{\mathbf{k}^m_l}$, and the detected target position, $\mathbf{k}^w_l$.
In each detection cycle the control rule and the physics engine are iterated several times[14] for all key-points. This leads to an exponential convergence of nodes corresponding to detected markers being moved towards the observed positions, while other nodes are interpolated smoothly and in a physically plausible manner by the constraints of the underlying physical model. By altering the controller gain, λ, the responsiveness of the tracking system can be adapted. While higher values make the tracking faster but very sensitive to single detection outliers, a smaller gain leads to a noticeable tracking latency. λ was manually optimized to a value around 0.9, which provided a good trade-off between theses two competing factors.

Model Properties and Parameters

In comparison to the mathematical model, the physical model is linked more directly to the model requirements formulated in Section 4.3.1. Distance preservation is directly controlled by the stiffness of links connecting adjacent grid nodes. The link stiffness can also be adapted for every link separately, allowing an inhomogeneous stretching of the paper to be modeled. It is important to underline, that the local stretching is not just a modeling artifact here. Instead, it is an important property, which is required for a numerically stable modeling of common paper deformations. The

[12] When using the real paper, the unprintable printer margin made it necessary to adapt the size of model cells adjacent to the model edges. However, for simplification, this fact is disregarded in the following.
[13] In the case of the physical model, only marker centers were used as key-points.
[14] In the experiments the iteration count was 10

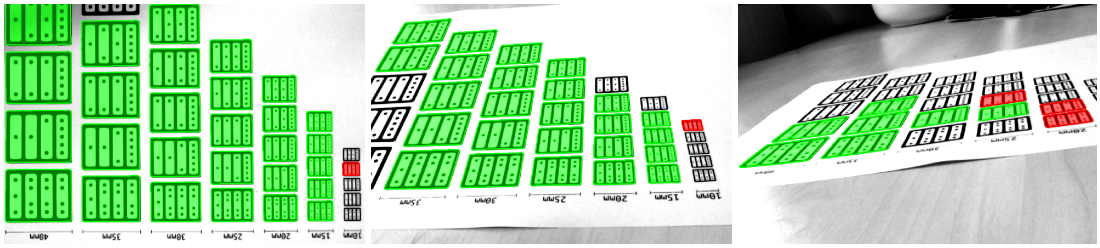
model curvature is limited by the stiffness parameter of links that connect non-adjacent nodes. Setting the stiffness to values close to zero even allows for the modeling of hard folds along existing edges of the 2D model grid. However, as the paper is only bent, but not folded, in the *picking-up* experiment, the use of a global stiffness value was sufficient. An extended model that is able to model folds is presented in Section 5.3. Another parameter that needs to be taken into account is the number of links to use. This was decided upon by adding links between all model node pairs $(n_{\mathbf{p}^m}, n_{\mathbf{q}^m})$, whose city-block distance,

$$d_{\mathbf{pq}} = \|\mathbf{p}^m - \mathbf{q}^m\|_{L1},$$

in the model space P' is less than or equal to $D_{\text{max}} \in \mathbb{N}$. Increasing D_{max} leads to a stiffer surface, which has to be compensated by lowering the global link stiffness. However, the real-time performance is also significantly decreased if too many links are used. In the final system D_{max} was set to a minimum of 4, since lower values did not allow the desired surface stiffness to be reached, which results in a model behaving more like cloth than paper.
The proportional gain λ and the number of physics iterations performed in each detection cycle are parameters that can be set by trial and error and through experience by visual inspection of the model. This allows the temporal responsiveness of the model to be adjusted. Smaller updates using lower values of λ, in combination with a higher number of physics iteration cycles, usually lead to a more realistic tracking behavior. However, in a real-time tracking system, the system performance scales linearly with the number of physics iterations and this can lead to performance issues if the iteration count is too high. Taking this into account, the number of iterations was set to 10. The underlying physics engine is also used to automatically provide self-collision detection. This helps avoid unrealistic model configurations in which parts of the model intersect each other. By inserting a ground-plane object into the physics scene, physical collision and contact simulation improves the modeling of the parts of the paper that are not detected by the vision system.

4.4 Evaluation

In this section the accuracy of the marker detection module and the presented modeling strategies are both evaluated qualitatively and quantitatively. The capabilities of the marker detection framework are systematically plotted using artificially rendered images. This means that the test results can be linked not only to image-space related quantities, such as a marker's pixel count, but also to real-world-related quantities such as *marker to camera distance* and *marker to camera elevation angle*.
Regarding the evaluation of the modeling framework, two different versions of the mathematical model were tested. A simpler one that uses three polynomial basis functions to model surface patches by planes (the *poly-3*-model) and a more complex one, that uses six basis functions including squared and mixed polynomial terms (the *poly-6*-model).
Even though the qualitative comparison (see Section 4.4.2) yields helpful insights into the capabilities and the drawbacks of the different models, some quantitative data was also collected. However, since there is no easy way to acquire reliable ground-truth information, artificial input data was used. For this, a GUI-editor was implemented that allows for mouse-gesture based interaction with a virtual model of the paper. We note that the editor does not use the paper models presented in this Chapter, but a more sophisticated extension of the physical model (see Section 6.1). In order to use the virtual paper model as an input for the modeling framework, the editor was endowed with an interface to simulate marker-based key-point detection results.
After comparing the course of the mean modeling error in the artificial interaction sequence (see Section 4.4.3), a few special cases are evaluated. First, localized error maps were created along a set of key-frames of the interaction sequence. These give more detailed information about where



(a) elevation 90° (b) elevation 45° (c) elevation 15°

Figure 4.10: Qualitative impression of the detection accuracy with respect to the marker's size and tilt angle using images of size 800×600 and an approximate distance to the markers of 300mm. Green markers were detected correctly, red markers were detected, but their ID was estimated wrongly. While the detection rate is optimal in case of a *frontal* view (a), a higher tilt angle leads to less positive detections and more ID-errors (b,c).

the particular models are prone to large errors (see Figure 4.15). As the major drawback of the simple mathematical model turned out to be missing distance preservation, this aspect is additionally evaluated in Section 4.4.4.

4.4.1 Fiducial Marker Detection Accuracy

In order to quantitatively asses the accuracy of the different approaches, the new fiducial marker design and the corresponding detection framework was evaluated using artificially rendered images. Given a fixed camera resolution, the main factors involved in reliable marker identification and 6D pose estimation are the distance between the marker and the camera and the elevation angle of the camera from the marker plane. While marker identification is at its most reliable when the marker plane is parallel to the camera plane (camera elevation 90°), the pose estimation is, due to pose-ambiguity [Schweighofer and Pinz, 2006], more accurate if the marker plane is tilted away from the camera plane (elevation $\approx 45^\circ$).
As a first step, the marker detection and identification mechanism was qualitatively evaluated and visualized (see Figure 4.10). For the tests, markers of different sizes were detected with different camera elevation angles in real camera images of SVGA (800×600) resolution. When the markers are viewed at an angle of 90°, robust detection of 15mm markers is possible (see Figure 4.10a). Average distortions of the marker quadrangles, arising from elevation angles of about 45°, make the detection worse due to a perspective-related decreased size of the markers (see Figure 4.10b). If the camera elevation angle is decreased even more (see Figure 4.10c), only very large markers can still be detected robustly.

Quantitative Evaluation of the Marker Identification Reliability

In order to provide quantitative results, artificially rendered input images were used. This does not only provide ground truth information in terms of position and orientation of the markers with respect to the camera frame, but it also allows for systematically sampling a large set of possible configurations. The sampling was performed on three parameters: The distance between the camera and marker center (in the range $[100\text{mm}, 900\text{mm}]$), the elevation angle of the camera normal from the marker plane and the rotation of the marker in the marker plane (both in the range $[0^\circ, 90^\circ]$). All other camera parameters were fixed, the camera resolution was set to 800×600. For each configuration, four different cases were possible:

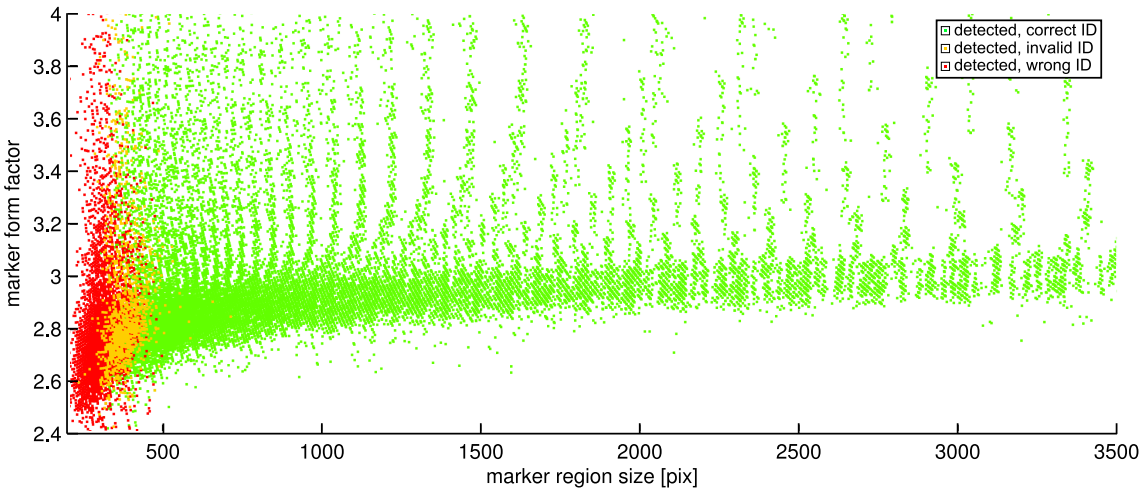


Figure 4.11: Quantitative evaluation containing 66000 samples from images of size 800×600 using a square marker with an edge length 30mm. The plot visualizes the detection accuracy with respect to image based features: marker region size and form factor. The structure within the green dots originates from systematic sampling of the marker-to-camera configuration.

1. *The marker is correctly detected.* This is the optimal case, in which a set of 2D-3D correspondences is made available for the modeling framework.
2. *The marker is not detected.* This results in missing key-points on the paper surface. The corresponding patch of the paper surface needs to be interpolated by the modeling mechanism.
3. *The marker is detected, but an invalid ID is extracted.* In this case, the marker ID selection heuristic allows the system to detect the marker identification error, since the resulting marker ID is not in the list of valid IDs. This means that this case is identical to case 2.
4. *The marker is detected, but a wrong yet valid ID is extracted.* This is the worst as wrong 2D-3D correspondences will be fed into the modeling framework.

The first evaluation was conducted with respect to the size of the marker in the image and its elevation angle with respect to the camera, approximated by the marker region's form-factor[15] (see Figure 4.11). Since these features are related to the marker projected to the image space only, the plot is independent of camera parameters. It shows that markers that are smaller than 500 pixels in size, cannot be detected reliably. The ID selection heuristic presented in Section 4.2.1 can filter out most errors if the marker size is not smaller than 400 pixels. Due to the sampling of the parameter space, which only indirectly results in a sample's x/y-coordinate in the plot, the relationship between the marker detection reliability and the form-factor is not obvious.

A second evaluation of how reliable marker detection is, was performed with respect to real world quantities (see Figure 4.12). The plot shows that the furthest possible distance between marker and camera is about 600mm, but only when the elevation angle is about $90°$. As the marker plane's elevation angle decreases, the distance at which marker detection is possible reduces significantly. In the vicinity of the maximum detection distance, errors are very likely to occur. Only some of these are filtered out by the ID-selection heuristic.
These results led to the conclusion that further heuristical constraints had to be incorporated to avoid system instability due to the many possible ID mismatches. The plot in Figure 4.11 revealed a possible template for this. Before an image region is used as a potential candidate for a marker's

[15] Defined by $C^2/(4\pi A)$, where C is the marker region's boundary length and A is it's area

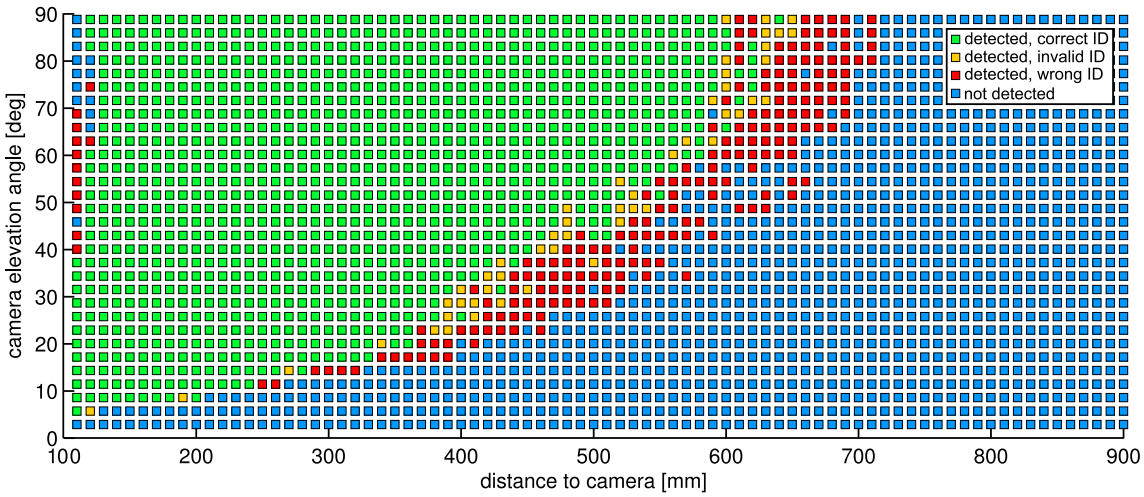
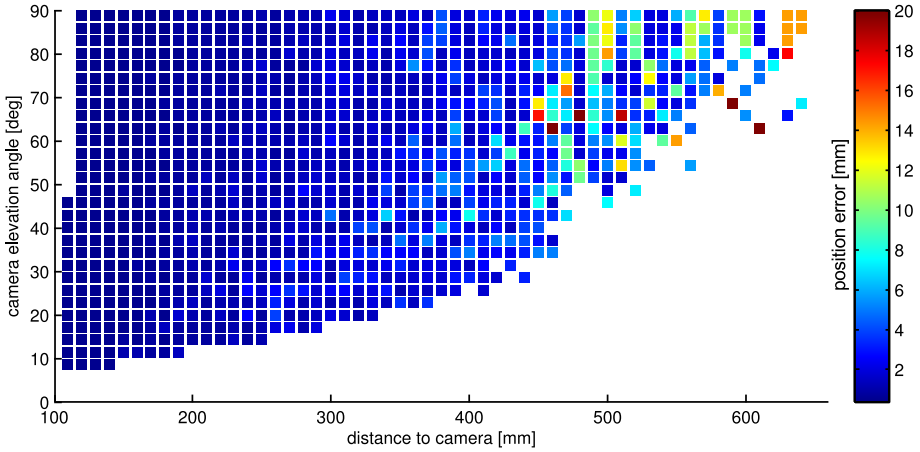


Figure 4.12: Dependency of the marker detection and identification accuracy with respect to the distance of the marker to the camera and the camera's elevation angle from the marker plane. The in-plane marker rotation was set to a representative, but fixed value of $30°$. The colors encode the detection and identification outcomes. Red samples are particularly bad, as they correspond to cases in which wrong IDs are computed but not excluded by the ID selection heuristic.

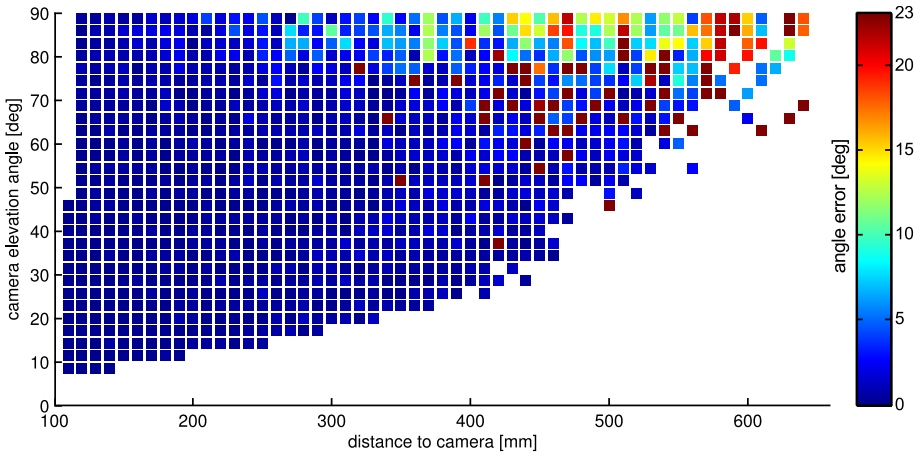
top level region, A, its size and its form-factor should be validated. The figure shows that a *minimum size threshold* of about 500 pixels avoids most detection errors. Furthermore, the sample distribution provides the information that the form-factor of marker regions is most likely within the range $[2.6, 3.8]$. Adding these constraints not helps to avoid detection and identification errors, but also yields a performance speedup by avoiding the graph-matching step for the regions filtered out.
It is worth mentioning that the results presented here strongly depend on the intrinsic camera parameters, in particular, the camera resolution and the focal-length. Increasing one of these directly increases the maximum detection distance. In addition, when working with real camera images, it must be taken into account that due to image noise and distortion the results will be worse than the presented results obtained from artificially rendered images.

Marker Pose Estimation Accuracy

Finally, the pose estimation accuracy was evaluated. Due to the fact that single camera pose estimation from a planar target is known to be much more prone to errors than multi-view-based pose estimation techniques, the evaluation is only focused on the single camera case. The evaluation was conducted in a similar fashion to the previous one, however, rather than plotting a discrete detection and identification outcome, the pose estimation error was split into position and rotation part and mapped to a pseudo-color scale (see Figure 4.13). While the position estimation error (see Figure 4.13a) mainly depended on the distance of a marker to the camera, the rotation estimation accuracy (see Figure 4.13b) was mainly linked to the camera elevation angle. In the marker detection pipeline (see Figure 4.4), the pre-processing threshold operation leads to 1-pixel noise at the borders of marker regions, which causes errors in the computation of the region's centers of gravity. As in the pose estimation step, a constant pixel shift of a keypoint leads to a higher position offset the further an object is away from the camera, the resulting errors are inversely proportional to the region's size. The errors in the rotation estimation are particularly high due to the ambiguity arising from an almost frontal view of the marker (i.e. where the camera elevation is close to $90°$). In these cases, the isometric perspective does not yield significant distortion, resulting in the estimation of orientations that are mirrored along the camera normal. This effect



(a) position estimation error [mm](clipped to 20mm)



(b) rotation estimation error [deg](clipped to 23°)

Figure 4.13: Pose estimation error, split into position and rotation errors. Empty cells correspond to configurations in which the marker could not be detected. **(a)** The position error mainly depends on the distance of the marker to the camera. In contrast, the rotation error **(b)** depends more strongly on the camera elevation angle. Here, a larger distance to the camera leads to a higher likelihood for extreme errors (dark red) that occur if the estimated marker tilt angle with respect to the camera normal is mirrored along the camera normal.

is compensated by a very short distance to the camera, which magnifies the camera's perspective distortion. This ambiguity is explained more detailed by Schweighofer and Pinz [2006].

4.4.2 Qualitative Comparison of Modeling Performance

In order to provide a coarse understanding of the modeling capabilities of the different approaches, a qualitative comparison was conducted. For this, a set of seven exemplary paper configurations was created and provided as input to each model (see Figure 4.14). Due to the fact, that the physical model implicitly performs temporal tracking, the configurations shown in the different images were always arrived at from an initial situation in which the paper was flat on the table. In addition, the support plane (known from the camera calibration) was explicitly added to the physics engine.

The first three rows in Figure 4.14 demonstrate the modeling capabilities in the absence of severe occlusions and deformation. Here, all three models perform very well. However, due to the

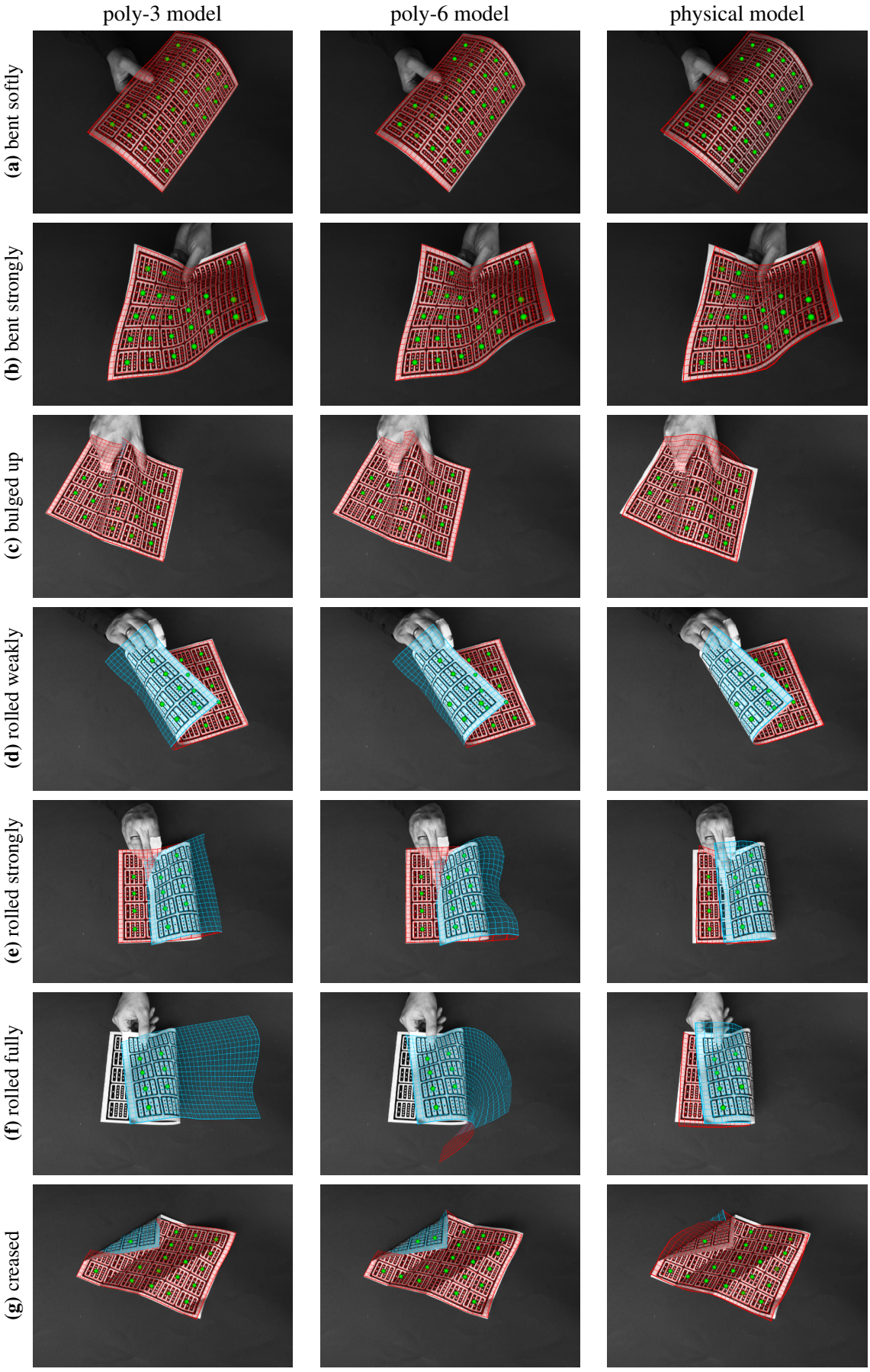


Figure 4.14: Comparison of the different paper models. Green dots mark the centers of detected markers.

fact that the physical model uses marker centers only, while the polynomial models also exploit the knowledge about the 3D orientation of the markers, the polynomial models have a better resolution in regions with higher local curvature. This effect can be seen in the right-most image of the *bulged up* case, when the physical model misses the paper bulge closer to the hand. As soon as it comes to heavier occlusions the physical model usually outperforms the polynomial models significantly. In particular, the weak distance preservation of the polynomial models can leads to very unrealistic modeling results (Figure 4.14d-e). The advantage of the physical model becomes even more obvious in the *rolled fully* example. Here, only markers on the paper part held by the hand are fully visible and detectable. Due to the absence of temporal tracking, the polynomial models give a very weak estimate of the lower paper layer. While the poly-6 model extrapolates at least the coarse trend of the paper, the poly-3 model cannot distinguish the depicted situation in which the paper is lying flat. In contrast to this, the physical model manages the configuration very well by using temporal tracking and by modeling collision and friction with the ground plane. The last row of Figure 4.14 compares the capabilities of the models in presence of creases. Here, the polynomial models perform better because the physical model is not set up to model extreme local curvatures or hard folds.
In summary, the qualitative comparison shows that the physical model behaves more realistically as soon as situations arise with anything more than minimal occlusions. In contrast, if fold lines are to be modeled, an extension that locally relaxes the maximum model curvature limitation of the physical model needs to be added to allow the physical model to compete with the polynomial models.

4.4.3 Quantitative Evaluation of the Mean Modeling Error

In the following evaluation, quantitative error data is presented. Due to fact that there is basically no way to get reliable ground-truth data from the actual paper configuration, artificial input data was used. The input data was generated by creating an interactive GUI-based editor that uses a more sophisticated physical paper-model presented in Section 6.1. The interaction sequence consists of 612 frames in which the paper is bent, turned around and even creased (see Figure 4.15). Even though it would have been possible to render the virtual interaction sequence in several virtual cameras in order to provide the rendered images as input for the marker detection framework, a more direct interface was implemented, where the position of detected marker regions was directly computed from the input model configuration. If a textured paper model were rendered, several rendering effects such as aliasing or clipping artifacts could have introduced too many uncontrollable error sources. The used interface allows for an easier and more direct control of determining which artificial markers are assumed to be detected by the vision framework. We used this feature to simulate two cases: in the first case, (see Figure 4.15 left graph), we modeled self occlusions by passing only such key-point correspondences to the models that belonged to markers that would theoretically have been visible from above. In the second case, (see Figure 4.15 right graph), self occlusions were not modeled, i.e. all key-point correspondences were used without taking into account their theoretical detectability.

Modeling Error

The evaluation is based on the local modeling error. Given the ground-truth paper function $p^* : P \rightarrow \mathbb{R}^3$, the local model error $e_{\mathrm{m}}(\mathbf{x}^m)$ of a paper model p (at paper coordinates $\mathbf{x}^m$) is given by the Euclidean distance

$$e_{\mathrm{m}}(\mathbf{x}^m) = \|p^*(\mathbf{x}^m) - p(\mathbf{x}^m)\|,$$

leading to the mean modeling error

$$E_{\mathrm{m}} = \frac{1}{WH} \int_P e_{\mathrm{m}}(\mathbf{x}^m) d\mathbf{x}^m.$$

For the calculations in Figure 4.15, E_{m} is approximated by a mm-sampling accuracy:

$$\hat{E}_{\mathrm{m}} = \frac{1}{210 \cdot 297} \sum_{x=1}^{210} \sum_{y=1}^{297} e_{\mathrm{m}}((x, y)^\tau).$$

Results

The course of the mean modeling error $\hat{E}_{\mathrm{m}}$ shows that the models have different strengths and weaknesses (see Figure 4.15). In the case of fast paper movements, the temporal tracking performed by the physics model sometimes leads to situations, in which the model is a few frames *behind* the input data yielding oscillating error curves (frames 200-300 and 450-500). In the case of strong deformation with high local curvature (key-frames 145, 410-566), the physics model's curvature limitation produces larger errors as well. In contrast, the qualitative evaluation underlines the behavior in case of severe occlusion such as shown in key-frame 104 and 234. However, while the physics model outperforms both polynomial models in frame 104, its mean error is still worse in frame 234, which can be explained by its inability to model the *wavy* structure of the right paper part. This can again be explained by the fact that the physical model uses only a single 3D input point per marker, i.e. a maximum of 30 2D-3D correspondences, while the polynomial models regard up 19 correspondences per marker.
The *poly-3* and *poly-6* models perform comparably for most of the trial. In some cases, the poly-6 model manages to mirror the detected surface structure extremely well (key-frame 234 right part of the paper). However its extrapolation abilities can also lead to very unrealistic modeling results, such as in key-frame 234 and 566.
The expected outcome that the physical model strongly outperforms the mathematical one could not convincingly be shown in this evaluation. This can be explained by the important fact that the simulation did not exclude markers that would have been occluded by the manipulating hands. In situations in which the visual perception is distinguished by severe occlusions (e.g. key-frame 104), the physical model performed much better. Therefore, in real manipulation scenarios in which the paper is not only occluded by itself, but also by the manipulating hands, we made the reasonable assumption that the physical model would dramatically outperform the mathematical one. Furthermore, the physical model implicitly performs temporal tracking and automatically interpolates between movements that originate from the physical simulation and such movements that are induced by the updates from the vision system. The performance of our final modeling and tracking system, which handles complex paper deformations including folds and crunches, proves this claim convincingly (see Section 6.2).

4.4.4 Distance Preservation Error

While the physical model uses a constraint-based built-in mechanism to provide distance preservation, the mathematical model does this only implicitly. Each local polynomial $\mathbf{P}_i$ implicitly minimizes the error in the input domain due to the polynomial regression based optimization in an algebraic least-square manner. In contrast, the soft-max interpolation mechanism smoothes the surface using the global smoothing parameter σ without regard to the distance preservation aspects. The surface distance between two marker centers heavily depends on the choice of σ.
In order to examine the quality of the distance preservation the tracking system was enabled to

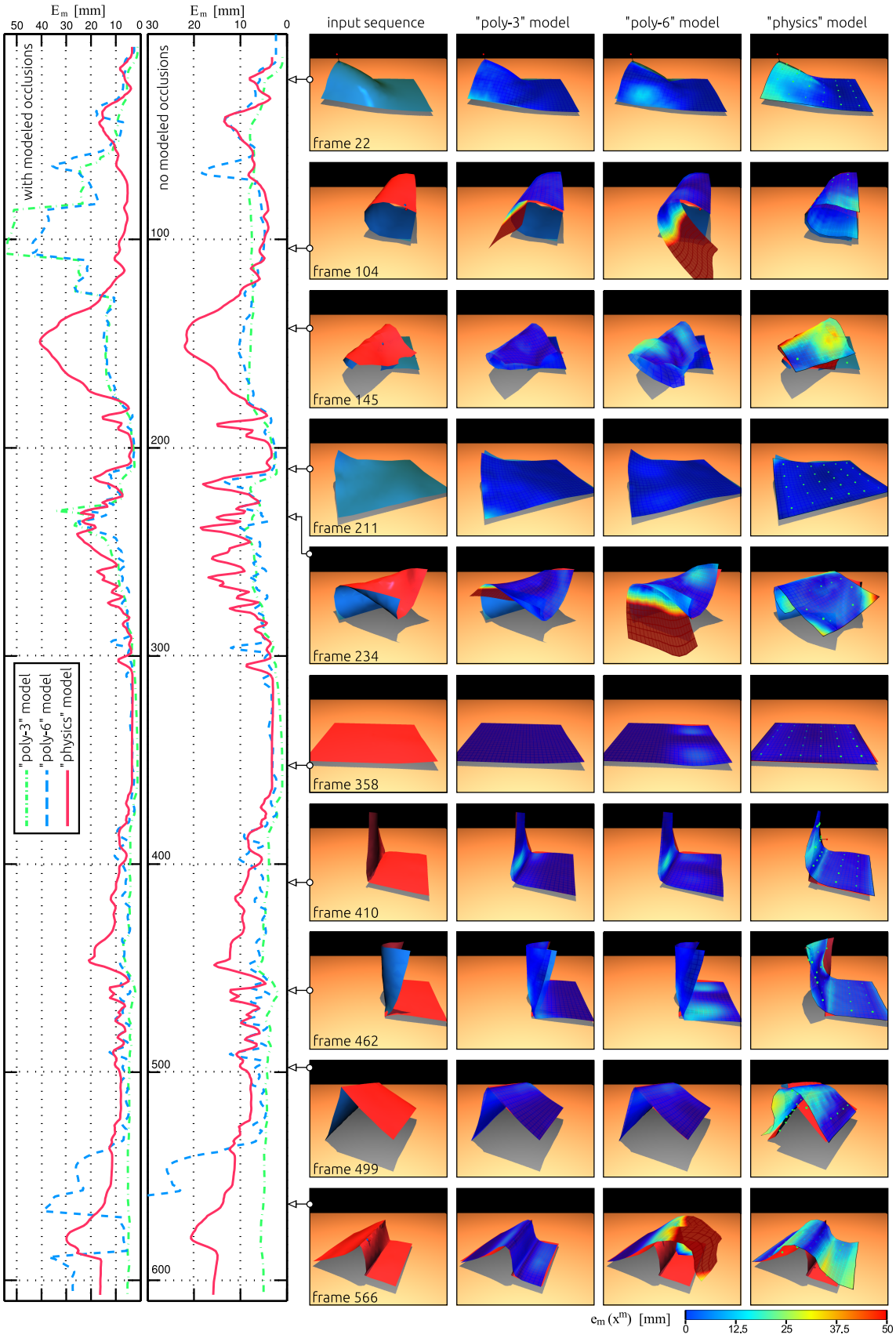
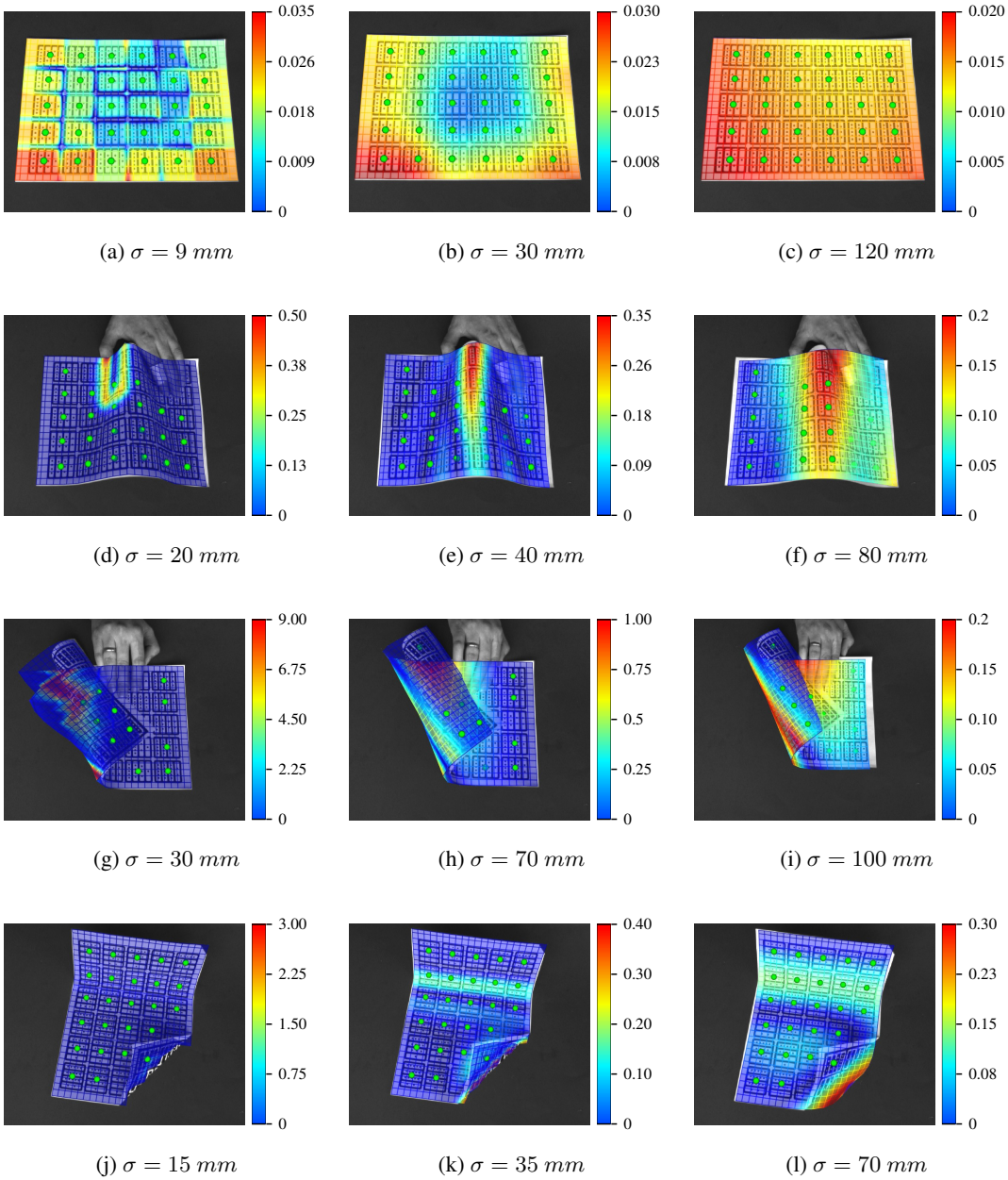


Figure 4.15: Course of the mean modeling error of the three different models in an artificial manipulation sequence (smoothed by a 3-frames running mean filter). In the left graph, self occlusions were modeled by disregarding key-points not visible from the top view (the images correspond to this graph). In the right graph, perfect detection was assumed, so each model was trained as if all markers were always detected. The images visualize the local error, $e_{\mathrm{m}}(\mathbf{x}^m)$, mapped to the model surface for the selected key-frames with modeled occlusion.



(a) $\sigma = 9\ mm$ (b) $\sigma = 30\ mm$ (c) $\sigma = 120\ mm$

(d) $\sigma = 20\ mm$ (e) $\sigma = 40\ mm$ (f) $\sigma = 80\ mm$

(g) $\sigma = 30\ mm$ (h) $\sigma = 70\ mm$ (i) $\sigma = 100\ mm$

(j) $\sigma = 15\ mm$ (k) $\sigma = 35\ mm$ (l) $\sigma = 70\ mm$

Figure 4.16: The Effect of the interpolation parameter σ for the distance preservation of the *poly-3* model. In the images the local distance preservation error, $\hat{e}_{\text{dp}}$ (in mm units), is mapped to the model surface. Green dots mark detected fiducial markers. **(a-c)** For the case of a flat model the error is very small. Note that the pseudo-color scale is adapted separately for each image. A higher interpolation bandwidth smoothes the error map. **(d-f)** If the paper is bulged up, a smaller σ allows for more precise modeling and the distance preservation error is very localized. While a high bandwidth lowers the maximum error, high curvature parts are smoothed too much. **(g-i)** For the case of stronger deformation a low σ makes the model overshoot the real curvature, which leads to very large errors. **(j-l)** Only a small σ can realistically model folds. If a higher σ is used, folds are smoothed too much.

augment the modeling result with a pseudo-color map that shows the local distance preservation error $e_{\text{dp}}(\mathbf{x}^m)$. The unit is set to be a mm and is obtained by walking small steps of $\Delta s = 1mm$ into each direction $\hat{\mathbf{e}}_1 = (1,0)^\tau$ and $\hat{\mathbf{e}}_2 = (0,1)^\tau$ of the model's surface. According to Equation 4.2, the resulting movements in 3D space should also be approximately 1 mm. Therefore, the estimated distance preservation error is defined by

$$\hat{e}_{\text{dp}}(\mathbf{x}^m) = \text{abs}(\|p(\mathbf{x}^m) - p(\mathbf{x}^m + \Delta s\hat{\mathbf{e}}_1)\| + \|p(\mathbf{x}^m) - p(\mathbf{x}^m + \Delta s\hat{\mathbf{e}}_2)\| - 2) \tag{4.7}$$

An evaluation of the distance preservation error for the *poly-3* model is given in Figure 4.16. The *poly-6* model produced similar results.

4.4.5 Conclusion

The evaluation shows that both presented approaches are well suited to model a deformed paper manifold. Even though the polynomial model performed slightly better than the physics model on average, the physics model was used for the *picking-up paper* experiment. The main reason for this choice is the better extensibility of the physics model and its much more realistic behavior in presence of severe occlusions, which occur frequently during interaction. Using a physics engine also allows additional features such as fold lines or folds on the basis of a physical description to be modeled. In contrast, extending the mathematical model requires a much deeper understanding of complex differential geometry methods. Particularly with regard to real-time capabilities, this not only requires complex tasks to be solved theoretically, but also imposes a major challenge if an efficient and real-time-capable implementation is desired.
As was mentioned before, the evaluation was also biased towards the polynomial models because these were able to use many more 2D-3D correspondences per frame than the physics model. The main reason for this is the lack of a model control-law that allows for controlling arbitrary model positions rather than single nodes only. Such an extension of the control law presented in this chapter can be found in Section 5.3.2.
During the course of this thesis, several iterations in which the physics model is extended are presented, resulting in a final version that can model fold lines and also memorize local deformations. In addition, new control laws are defined that allow for arbitrary paper coordinates to be controlled.

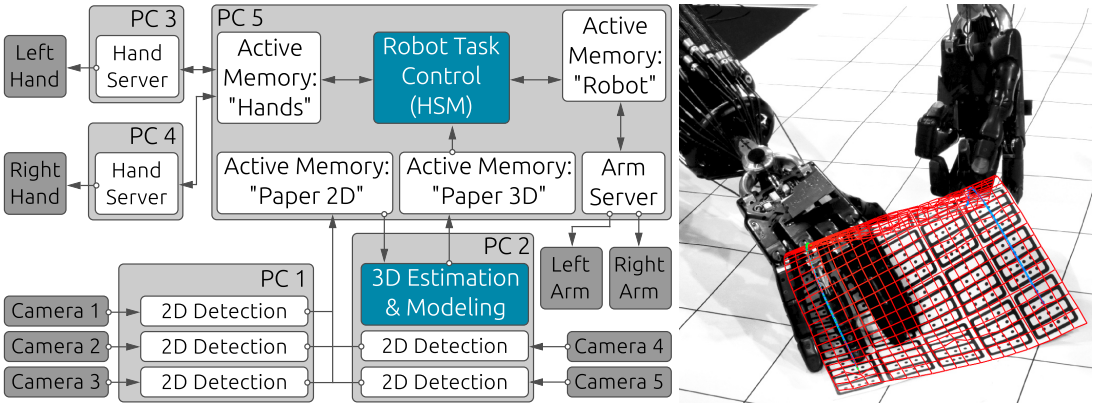
4.5 Robot Control

Using the presented paper tracking and modeling system, an interaction sequence was implemented that enables a bi-manual robot to pick up a sheet of paper lying on a flat surface[16]. Picking up the paper bi-manually is easier to realize since a more human-like strategy in which one thumb is carefully brought under a corner of the paper would require extremely sensitive tactile feedback (see Figure 4.1). Nonetheless the implemented bi-manual strategy mirrors how a person, given a certain situation, might pick up the paper (see Figure 4.17b).

4.5.1 Vision- and Robot System

For the *picking up* experiment, four separate Active Memory instances were created to facilitate inter-process communication (see Figure 4.17a). Five cameras were connected to two separate PCs

[16] A video of the robotic picking up sequence can be found in [Elbrechter et al., 2011b].



(a) IPC Scheme

(b) Robot picking-up paper

Figure 4.17: **(a)** Data flow and inter-process communication (IPC) structure of the setup. IPC is implemented using four event driven *Active Memory* instances. For each camera a dedicated image capturing and 2D marker detection process sends its results to the *Paper 2D* Active Memory instance. The *3D Estimation and Modeling* component processes the 2D marker positions into a 3D model of the paper surface, which is committed to the *Paper 3D* Active Memory instance. The *Robot Task Control (HSM)* schedules and monitors the action sequence carried out by the bi-manual robot system to pick up the sheet of paper in a data-driven fashion. **(b)** Image of robot picking-up the paper bi-manually with overlayed model.

to distribute the CPU-load and optimally exploit the available Fire-Wire bandwidth. Each camera image was processed by a dedicated 2D paper-detection process. The results were sent to a *Paper 2D* Active Memory instance. In order to avoid expensive XML-parsing, the positions of detected marker-regions were sent as binary attachments of simple XML-headers that define a minimum set of meta data, such as the associated camera ID. The *3D Estimation & Modeling* process subscribed to these documents. The last available frames of each camera were used to compute 3D positions of the detected marker regions. In order to avoid triggering and synchronization issues of the 2D detection processes, the paper was simply assumed to move *slow enough* to perform this in an asynchronous fashion. This also allows the 3D-detection and modeling frame-rate to be decoupled from the camera frame-rate which was, due to the bandwidth of the Fire-Wire interfaces, limited to 15Hz. The estimated 3D positions of marker-region are fed into the modeling framework which computes the paper model using one of the presented modeling approaches. The resulting model, parametrized as a regular grid of connected nodes, is sent to the *Paper 3D* Active Memory instance. In addition, some task-related meta features, such as the papers center-coordinate frame and the position and orientation of its highest bulge, are also sent. These could also be computed by the *task control* unit, but due to performance and practical reasons it proved better to implement this withing the *3D Estimation & Modeling* process.

The robot is logically controlled by a hierarchical state machine (HSM) [Harel, 1987] that is specified in XML-syntax. Dependent on the current state, the HSM controls both robot hands and robot arms. Robot control is also realized by Active Memory instances. These are used bi-directionally, i.e. the task HSM sends robot-movement commands interpreted by the arm and hand server processes, which in turn provide feedback of the current robot sensor information. In addition, a query mechanism is provided, that allows for interacting with the robot-servers internal scene representation.

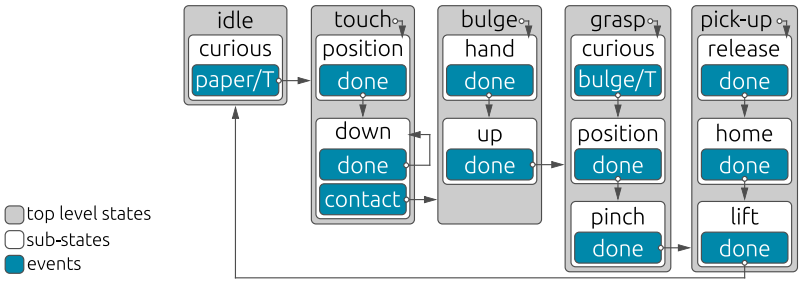


Figure 4.18: Hierarchical state machine (HSM) used for robot control in the picking-up paper experiment

4.5.2 Picking up Paper With the Robot

The final system that was built to bi-manually pick up a sheet of paper lying on a flat surface is depicted by the *Robot Task Control* HSM (see Figure 4.18). Even though internally an almost linear state-order is implemented, the HSM is used as an overarching framework. The system starts in an *idle* state and then can be manually put into the *idle:curious* sub-state, in which it awaits information from the modeling unit regarding the paper coordinate frame (*paper/T*). The transformation is simply derived from the mean marker transform. Once such an event is received, the *touch* state is entered, which directly transitions into the *touch:position* sub-state. Here, the robot's right hand is actuated and in parallel the right arm is placed relative to the received paper frame *paper/T*. When both actions are finished, the *touch:down* sub-state is activated. The system loops, moving the right arm towards the paper-plane until contact is established. When the system was implemented, no built-in controller for contact-establishment was available. Therefore this was implemented as an active perceive-action loop that exits when the finger-displacement from the reference posture overshoots a given threshold. Once contact is established, the paper can be bulged. This is performed by exploiting the compliant design of the shadow hands. If the joints were completely stiff, moving up the arm and slightly closing the hand would have to be synchronized to avoid damaging the hand when closing too fast or loosing contact with the paper when moving up too fast. However, the compliance of the hand allows the final bulging hand posture to be simply actuated, which can, due to the resistance of the table-top, not be achieved in a single step. Therefore the hand constantly preserves contact while the arm is slowly moved up in the *bulge:up* sub-state.

When the paper is suitably bulged, it can be pinch-grasped by the left robot hand. Starting in the *grasp:curious* sub-state the system waits to receive the bulge's coordinate frame *bulge/T*, also computed by the modeling unit. For this, the elevation of the paper model's long edges is approximated by polynomials of degree five (see blue and green lines in Figure 4.19). The bulge's highest line is defined by the connection of the two maximums. In the *grasp:position* sub-state, the left hand is positioned relatively to *bulge/T* while the right hand actuates a pre-grasp posture for pinch-grasping. Subsequently, in the *grasp:pinch* sub-state, the right hand is moved relative to *bulge/T* in order to bring the bulges top-center into the center of the following pinch-grasp final posture.

Before the paper can be lifted, the right hand has to release the paper, which is performed in the *pick-up:release* sub-state. Here, the compliant behavior of the hands turned out to be actually a disadvantage. Even emptying the hand's muscles does not necessarily lead to a fully relaxed hand, but sometimes rips the paper from the left hand's pinch-grasp. Therefore, this step had to be optimized by releasing in several steps. First, the right hand is opened carefully, i.e. using a very low stiffness value leading to a very slow finger movement. After two seconds the arm is moved upwards while the hand muscles are simultaneously emptied. Finally, the right arm is moved away from the paper in the *pick-up:home* sub-state. In the final sub-state *pick-up:lift*, the sheet of paper held by the left hand can be lifted and the system returns to the *idle*-state. Real images of the

bi-manual picking-up sequence are presented in Figure 4.19.

4.6 Discussion

In this chapter a system was presented that enabled a robot to bi-manually pick up a standard A4 sheet of paper lying on a flat surface. The system was developed from the ground up, but used some existing frameworks for physical simulation and for the event driven low-level robot control and inter-process communication. The main components of the system are the marker-based visual detection of paper-key-points, the physics based modeling engine and the robot-control HSM.
The development of the final robotic interaction sequence confirmed the presumption that even picking-up paper is a complex task for an anthropomorphic robot. Even though a lot of effort was exerted optimizing the relative tool transforms necessary for the bulging and pinch-grasping, the final system only had a reliability of about 60%. The main reason for failure was dislodging the pinch-grasped paper when releasing the bulging right hand. A better mechanism to compensate for the problems with the hand's muscle tension is clearly needed. Furthermore, actual pinch-grasping was also implemented in a feed-forward fashion. The computation of the position and orientation of the grasp point is statically triggered, and then all the information is derived from a single frame. A more detailed analysis of an approximate paper shape, optimally taking several frames into account[17] would most likely help to make this more robust.

4.6.1 Visual Detection

The marker-based visual detection engine proved perfectly capable of providing a sufficient amount of information about the sheet of paper. However, the fact that the paper needs to be fully augmented by fiducial markers is disappointing. For the experiment presented here, a much simpler visual detection method could also have worked, but with the caveat that it would fail in more complex interactions. However, when complex interactions lead to severe occlusions or heavy deformations of the paper, such as iterated folding, even the presented marker detection framework might not work. Although the new marker layout explicitly introduces a mechanism to avoid false-positive marker detections and erroneous marker ID estimation, it turned out that the used combination of image-resolution and relative size of the markers is already exhausted. As soon as the paper is folded, many markers become undetectable because their region-structure is cut by occlusions or along the fold lines. This effect could be minimized by using more marker that are smaller, but for the present marker layout, this would require a higher image resolution to avoid an increase of detection errors. This in turn would, due to bus-limitations, reduce the possible frame-rate, which would entail further issues.
Therefore, to improve visual detection, two possible alternatives were examined. In Section 5.2.1 a new marker layout (similar to the one introduced by the ARToolKit Plus library) that allows markers to be smaller while preserving a tolerable error rate is presented. Taking into account negative criticism about the necessity for fiducial markers in the first place, an alternative 3D-point-cloud based detection engine and the necessary adaptions to the modeling engine are presented in the Sections 6.2 and 6.3.

[17] The physics engine internally performs physical tracking. However, an explicit averaging over several frames would still provide more stable results.

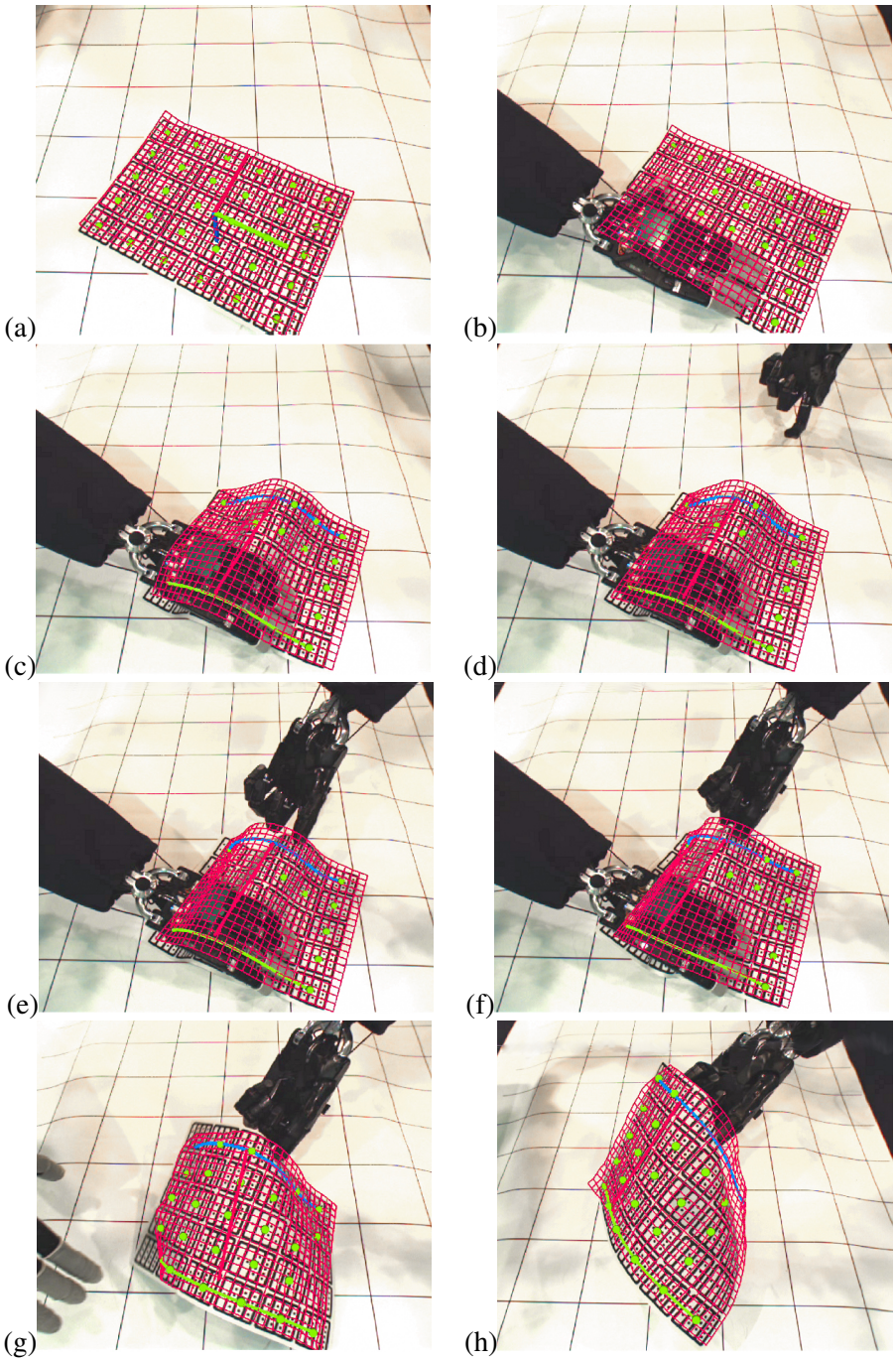


Figure 4.19: Images of the robotic interaction sequence using the physics based model (note, the *left* robot hand is seen on the right side of the image and vice versa). **(a)** The paper is put somewhere into the workspace. Due to the lack of occlusions, all markers (green dots) are detected and the paper model is perfectly fitted. The paper's center coordinate frame is computed and visualized. **(b)** Right hand established contact with the paper. **(c)** Right hand bulges up the sheet of paper. The long paper edges are approximated by polynomials (green and blue lines) and the connection between the maximums (thick red line) defines the position and orientation for the left hand's pinch-grasp. **(d)** Left hand approaches the paper actuating a pre-grasp posture. **(e)** Left hand is positioned. **(f)** Paper is pinch-grasped. The right hand *carefully* releases the paper. **(g)** Right hand moves back to home position. **(h)** The paper can be lifted by the left hand. The shape of the paper is modeled well during the whole interaction sequence.

4.6.2 Physics-based Modeling

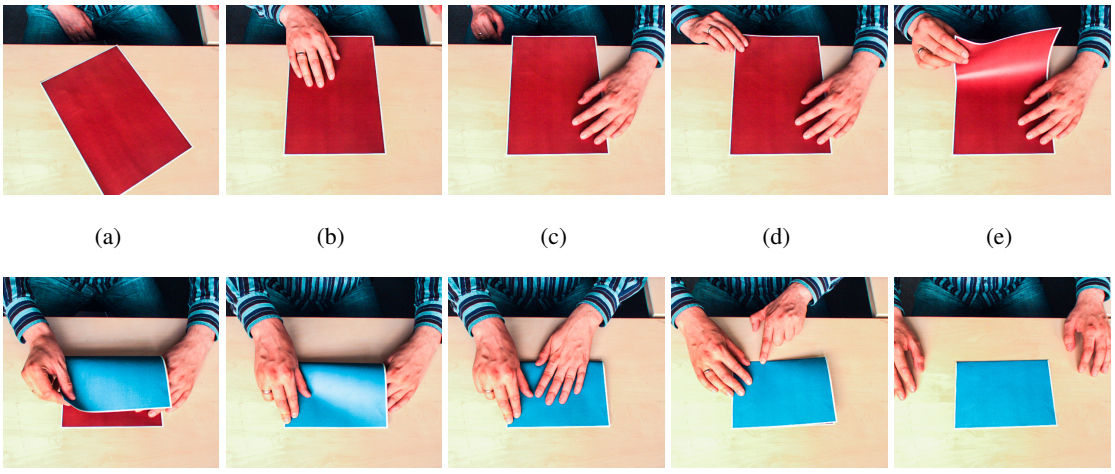
The physical modeling engine serves well for the picking-up task. Yet it could be argued that a much simpler modeling unit that provides the paper's center coordinate frame and the position and orientation of the top of the bulge would have been sufficient. However, if a more sophisticated manipulation is required to model more complex behavior of paper such as creasing, folding and memorizing deformation, a hard-coded model would definitely become too complex to maintain. It is important to emphasize that the used modeling framework is fully generic and for the model control, no task-specific information is used. Assuming the paper is to be bulged in advance would obviously provide additional constraints for the model deformation but also link the model-update mechanism strongly to the task.
Another major drawback of the physical modeling approach is the limited control law that only allows existing model nodes to be controlled. In order to avoid having to adapt the grid-structure of the model to reflect the layout of detectable keypoints, a more general control law would be required. This would already constitute a major prerequisite for the use of marker-less tracking methods based on generic keypoint detection. In order to model not only bending but also folding, the features of the existing bending constraints could be altered. Again, this can be reached by two different mechanisms. As a first rather simple extension, only the stiffness of existing bending constraints is adapted. A system, that implements this technique is presented in Section 5.3.1. Disproportionally more complex is an extension that automatically inserts new nodes along a fold-line. For this, the regular grid-structure of the paper-model would have to give way to a more general irregular triangle-based structure. Such a model, like that already used to generate artificial ground-truth-data in Section 4.4, is presented in Section 6.1.

4.6.3 Robot Control

Although the design of the presented robot control sequence is very specific to the picking-up task, its components, in particular the finger displacement based contact establishing mechanism, are likely to be reusable building blocks for further more sophisticated manipulation systems. In addition, the development of the system helped to get a better understanding of what difficulties can arise when trying to manipulate paper using the present anthropomorphic robot setup. While some of these difficulties, such as the sensitivity of the system when working with hard-coded relative transforms, were anticipated, some others, such as simply trying to release the sheet of paper, were not obvious in advance.
The major drawback of the system is the missing use of closed-loop tactile and visual feedback. The information from the visual tracking and modeling system is only used in a few key-frames to align the otherwise feed-forward manipulation sequence with the observation. In a simplest case, the system could be enhanced by installing a concurrent surveillance process that automatically detects non-tolerable differences between the present and an anticipated current state in order to stop the robot given an error. Building on this, a recovery mechanism could be implemented, which allows the system to undo recent steps until a recovery point is reached that allows further actions to be re-planned. This would also bring out a starting point for online-leaning by altering interaction parameters after an error has occurred. Regarding the use of tactile feedback, the integration of parametrizeable built-in controllers in the robot control framework that incorporate tactile and proprioceptive feedback would be of great help for the development of further interaction tasks. Two such controllers and their integration into the robot system are presented in Section 5.5.3

5 Bending and Folding

After endowing our robot with the ability to pick up a sheet of paper bi-manually, we wanted to understand the requirements of anthropomorphic robotic folding strategies, in order to obtain a deeper understanding of anthropomorphic robot manipulation in general. Recalling the variety of common manipulations that can be performed on and with paper, *folding* was chosen as a crucial ability the robot would need. However, after reviewing existing robotic paper folding systems (see Section 4.1), it became clear, that existing methods cannot be applied in the present scenario. While other systems use specially designed *folding robots* [Balkcom, 2004; Dubey and Dai, 2006; Tanaka et al., 2007], our task was to realize these actions on the presented anthropomorphic robot setup (see Section 2.1.1). The lack of a specialized robot with custom-designed tools for folding paper shifts the focus from the development of folding theorems and interaction sequences that lead to a certain complex folded structure to more fundamental questions. For example, the new system will have to deal with severe occlusions while the paper is being folded. Moreover, in order to compensate the lack of optimal calibration, feedback-based controllers are needed that use visual and haptic feedback to establish or maintain contact forces during interaction.
In order to get deeper insights into the complexity of paper folding using an anthropomorphic robot, we initially limited the scenario to *folding a piece of paper in half*. The folding robot presented by [Balkcom, 2004] (see Figure 5.2) would perform this by aligning the paper's central line with the fold slot using a vacuum gripper, followed by an automated folding step. Here, a blunt folding blade descends from above into the fold slot of the support plane. The paper, which intersects the slot, is pushed downwards by the blade with both ends of the paper getting bent upwards. When the blade is removed, the slot is closed to ensure the crease remains. Finally, the blade swipes over the opened fold slot from one side to the other in order to place the paper again flat on the support plane.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 5.1: Folding a sheet of paper in half. **(a-b)** shift paper towards table edge **(c)** fixate with left hand **(d)** grasp corner with right hand **(e-f)** bend over **(g)** align corners **(h)** crease coarsely **(h)** harden crease **(i)** folded sheet of paper

In contrast, an anthropomorphic robot needs to perform a human like sequence of actions. For demonstration, an equivalent folding sequence applied by a human is presented in Figure 5.1. In order to pinch-grasp a corner of the paper, it is shifted towards an edge of the support plane. Then, one side of the paper needs to be fixated with one hand, while the other hand pinch-grasps the corner to bend the paper in half. Once, two corners are aligned, the pinch-grasping hand can be used to fixate the aligned layers of the paper again. This allows the other hand to be released to make the fold. This is usually performed in two steps. First, a little force is used to flatten the paper around the target fold line with all fingers. After this, a precise one-finger creasing motion is applied to harden the crease.
Even though it would be possible to use it, the presented sequence does not incorporate the *picking-up paper*-sequence presented in Chapter 4. This is due to the fact that in the present scenario, an initial shifting of the paper is much more natural. Furthermore, the final robustness of the robotic picking-up sequence was not high enough to use it as a mandatory initial step for the robotics experiment conducted in this chapter.
In contrast to the picking-up experiment where the paper could be modeled by a smooth 2D surface in 3D space, folding paper requires folds to be explicitly modeled. This is implemented by altering the stiffness of the physical paper model's bending constraints along fold lines. In order to increase perception accuracy, the existing fiducial marker detection system was enhanced using a new set of fiducial markers. The new markers can be detected more robustly even if they are much smaller, allowing the possibility to print even a more detailed grid of markers on the paper.
This chapter is organized as follows. In Section 5.1 related work especially relevant for *paper folding* is presented and discussed. Subsequently, Section 5.2 introduces the new fiducial marker layout and the corresponding marker detection pipeline. The adaptions to the structure of the physical paper model is presented in Section 5.3. A generalizing extension of the model control law that moves the model according to the observation is also presented. Once perception, modeling and the connection between these has been established, the capabilities of the framework are evaluated by monitoring several human folding sequences (see Section 5.4). Finally, in Section 5.5, the system is used to implement the folding sequence on our anthropomorphic robot system. For this, a simplified inter-process-communication scheme and a new HSM for the robot control is presented, and a set of new visuo-haptic feedback-controllers are introduced. Finally, we conclude with a discussion in Section 5.6.

The work presented in this chapter is mainly based on the author's conference paper:

> **Folding Paper with Anthropomorphic Robot Hands using Real-Time Physics-Based Modeling** [Elbrechter et al., 2012a]

This paper won the *best paper award* at the IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012), in Osaka, Japan. An associated video can be found on the CITEC YouTube-channel [Elbrechter et al., 2012b].

5.1 Related Work

The robot paper folding experiment encompasses several sub-disciplines. Even though the resulting processing pipeline is very similar to the one developed for the picking-up experiment (see Section 4.5), a set of new aspects needs to be anchored in the literature. In contrast to the related work presented in Section 4.1, here particular focus is placed on the interaction.

5.1.1 Visual Detection

The visual detection of deformed and creased surfaces poses several, still unsolved, challenges. While rigid object tracking can be approached using RANSAC-optimized correspondences of locations of edges [Drummond and Cipolla, 2002] or generic image features [Miao et al., 2011] methods, deformation within the object leads to changing model coordinates of the features. Furthermore, the presence of severe occlusions caused by the manipulating hands as well as by the paper itself when it is folded, makes it very difficult to apply standard methods such as key-point or edge tracking.
It is for these reasons that visual detection of deformed and creased surfaces is usually facilitated by either using fiducial markers [Bersch et al., 2011; Mitani, 2006], or by restricting the amount of occlusion that is allowed. Kinoshita and Watanabe [2008] used edge features, but they manually selected well suited perception frames to avoid having to deal with occlusions (see also Section 4.1.2). The tracking and manipulation system presented by Schulman et al. [2013b], used RGBD-point cloud data to drive an internal physical model of a towel-like object while it was folded twice. In order to avoid the use of fiducial markers, they assumed trivial color based segmentation of object sub-patches. Furthermore, the manipulating human hands were filtered out explicitly by using gloves of the same color as the background. In the work presented by Balkcom and Mason [2008], visual feedback was omitted completely by insisting on a well defined initial location of the paper, resulting in only feed-forward manipulations being able to be carried out. In [Tanaka et al., 2007], a paper folding robot system used edge features to determine paper contact positions, however, since their edge detection was not reliable enough, edges had to be interactively labeled by a human assistant.
Due to these difficulties, visual detection of the sheet of paper here is again based on fiducial marker tracking (reviewed in detail in Section 4.1.1). The improved markers used for the folding experiment (see Section 4.2) are distinguished by a self-error-correcting 2D BCH code [Bose and Chaudhuri, 1960] at their center. Using an explicit model of the detected sheet of paper as an interface between recognition and robot control, the marker-based tracking module can easily be replaced by a more sophisticated marker-less tracking option. Such a system was developed and is later presented in Section 6.2.
In addition, we note that the visual detection engine presented in this chapter is not able to detect *when* and *where* the paper is folded automatically, entailing the necessity that fold lines must be added manually. The automatic detection of folds is a very difficult problem with its own set of issues and these are investigated in Section 6.4.

5.1.2 Modeling Foldable Objects

Objects that can be folded such as paper, cardboard boxes and bendable sheet metal are usually modeled as piecewise rigid objects (faces) connected by revolute joints (creases) [Liu and Dai, 2003; Lu and Akella, 2000; Song and Amato, 2004]. By not explicitly modeling bending, the model is restricted to a finite number of DOFs leading to a well defined configuration space $C = \mathbb{S}^{1^k}$, where $\mathbb{S}^1 = [0, 2\pi[$ is the unit circle and k is the number of joints. However, this leads to severe challenges if creases intersect [Balkcom and Mason, 2008]. In this case, the resulting kinematics system can no longer be treated as a kinematic tree. Instead, it has to be handled as a much more complex closed kinematic chain. Origami models (reviewed in Section 4.1.2) explicitly model creases. However, most approaches are only able to represent discrete 180-degree fold angles and are therefore not suited for real-time tracking, which requires continues folding angles to be possible. In contrast, it was important for our requirements that our model tracking system could represent arbitrary folds. Schulman et al. [2013b] presented a comparable robotic system, which was able to manipulate deformable objects, such as rope, cloth and foamed material.

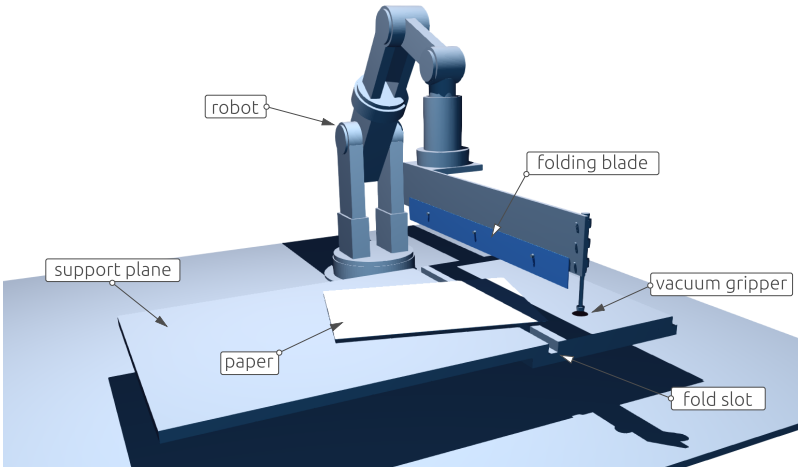


Figure 5.2: Author's impression from the video of the Balkcom's origami folding robot [Balkcom, 2004]. The robot is endowed with a custom-designed tool. For shifting the paper, a vacuum gripper is attached. Folding is applied by inserting the folding blade into the fold slot, followed by closing the slot while the blade is removed.

They also used the Bullet physics engine for modeling, but their system did not explicitly model folds. In comparison to paper, the objects they chose were all thick, so that folds could always be satisfactorily approximated by high curvature bends. In contrast, the modeling engine presented here explicitly models folds by adapting the stiffness-constraints of the physical model along folds lines.

5.1.3 Robot Control

In 2004, Balkcom [2004] introduced a custom designed origami folding robot (see Figure 5.2). The precise folding mechanism works without the need to fixate the paper. The limited capabilities of the robot, however, also illustrate the need for more general purpose robots like the one we are working with. Furthermore, due to the precision achieved by its custom design, it can fold paper in a feed-forward manner. Assuming two paper edges are initially aligned with the front edge of the support plane and the fold slot, no further visual or haptic feedback is needed. Another comparable robot system designed to fold paper was presented by [Tanaka et al., 2007]. In contrast to Balkcom's custom-designed paper folding robot, they used a *more dexterous* robot hand. Driven by the idea of *going one step further* than Balkcom, they designed a *minimal* robot able to fold an origami *tadpole*, which includes not only valley folds, but also a squash-fold, which was not possible on Balkcom's robot. However, even though their developed robot hand had four fingers with tips and nails, it's anatomy is too different from our anthropomorphic robot hand to allow for directly porting their folding approach. Due to the kinematics of their folding robot, folding can better be compared with two humans folding paper each holding a stick in both hands. Their work provides important insights into the core issues of dexterous robotic paper manipulation and includes very honest reporting of the low success rates and the long completion times of their system. Their idea is to perform several iterations in optimizing the robot hand in order to create more and more complex origami models. They expect that through an evolution-like optimization a final generation that is possibly even able to fold an origami crane[1] could look very similar to the human hand.

[1] The classical origami crane is considered to be much more difficult than the tadpole. While the tadpole folding can well be described in a 6-step instruction, instructions of folding the origami crane commonly have more then 15 steps.

5.1.4 Planning Folding Sequences

Another related field of research concerns planning approaches for folding sequences. In particular the automatic inference of robot control patterns for the folding of cardboard boxes is very important for the packaging industry. Here, foldable objects are commonly modeled as kinematic trees, comparable to robots, allowing the application of classical robot planning algorithms. Liu and Dai [2003] presented a planning approach for folding cardboard boxes with two one-finger robots. In their system, a *configuration control point* (CCP) is attached to the center of each face of the carton model. After computing an optimal folding sequence for the faces of the cardboard box, by heuristically modeling face motion and collision, these CCPs were used to generate motion trajectories for one-finger robots. A comparable system that also models cardboard boxes as articulated robots was presented by Lu and Akella [2000], who used fixtures for efficient folding. Song and Amato [2004] presented a motion planning approach for more complex kinematic chains, such as a polyhedral model of a cardboard football and even 120 DOF protein strands. Their system used *probabilistic roadmap methods* (PRMs) [Kavraki et al., 1996] to infer folding sequences from a given kinematic model.

In summary, it can be stated that none of the related systems employed anthropomorphic robot hands. Furthermore, all systems bypass the visual detection issues that arise from occlusions by either adapting the hardware to minimize occlusions, by using fiducial markers, or by pre-planning the whole manipulation sequence. In addition, none of the mentioned systems is able to model all three major aspects of paper deformation:

- Compliant, soft-body-like behavior
- Plastic behavior, leading to fold memorization
- Locally relaxed bending stiffness along straight fold lines

As already discussed in the introduction, further approaches to foldability, complexity classes of origami and planning of origami folding sequences deal with higher-level problems that are less relevant for the task of creating a single fold. However, since the presented formalisms only assume a robot capable of folding paper along a defined line, it is not overly optimistic to assume that they could be extended without major adaptions as soon as anthropomorphic robots are endowed with this ability.

5.2 Perception

After analyzing the weaknesses of the fiducial marker detection framework developed for the picking up experiment, it turned out that its major drawback was the large market size required for robust detection. In addition, the large minimal marker size makes the structure of markers very likely to be *broken* by occlusions, leading to even more missing detections. This effect is even amplified by folds that break all intersected markers apart. The used marker detection approach, based on topological image region containment information was too prone to detection errors. While not detecting a visible marker only leads to gaps in the dense key-point grid used for the modeling, false positive detections and wrong marker identifications involve major issues that could even lead to instability of the physical model. The ID selection heuristic that was introduced did not provide a sufficient improvement to this issue.
Therefore, another fiducial marker type, similar to ARToolKitPlus/ARTag markers (reviewed in Section 4.1.1) was added to the marker detection framework. The markers are distinguished by a rectangular black region that contains a 6×6 bit binary BCH code in its center, which is used to

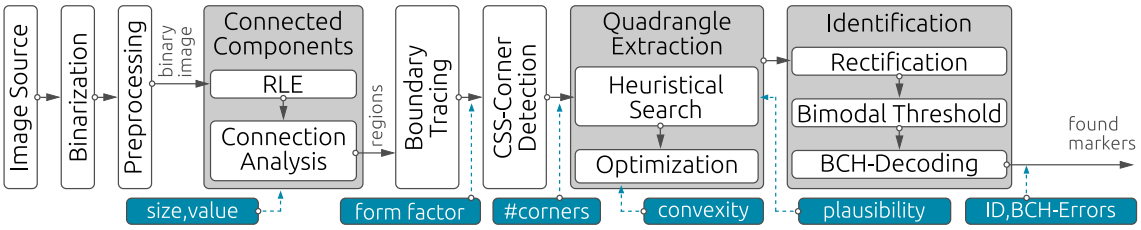


Figure 5.3: Tool-chain for the detection of BCH-code markers. Blue boxes illustrate how image regions are filtered. The binarized and de-noised input image is passed into the connected component module, in which the resulting image regions are internally filtered by size and value. The lengths of the remaining region's traced boundaries and the region's pixel count are then used for a form-factor-based filtering. The following *curvature scale space* (CSS) based corner detection approximates each region by a polygon serving as input for the quadrangle extraction module. Here, several heuristics are applied to find quadrangular region parts that are potential marker bases. After optimizing the quadrangle corners, convex quadrangles are perspectively undistorted by rectification, and their inner BCH-pattern is binarized and BCH-decoded.

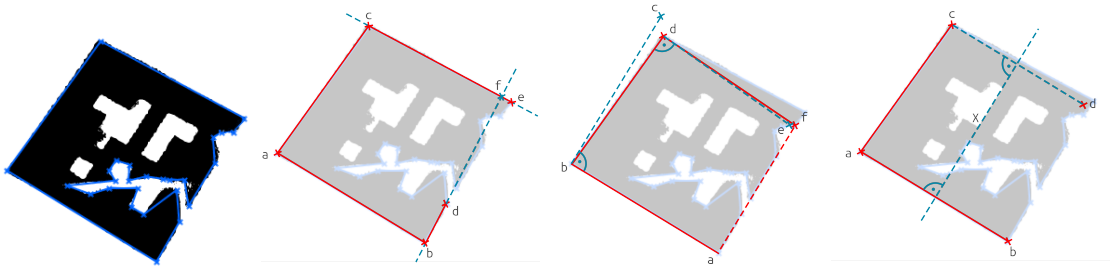
identify a 12 bit marker ID. The BCH code can automatically correct up to 4 bit-errors, however, it turned out, that dropping markers with an error count larger than 3 significantly increases the detection accuracy at the cost of only a minimally decreased recall. As discussed in Section 4.1.1, the existing fiducial detection libraries did not meet the requirements, in terms of speed, accuracy and licensing. Therefore, a marker detection plugin for BCH code markers was integrated into ICL's marker detection framework. The author emphasizes that only the detection pipeline for the BCH code markers was re-developed. The marker layout itself was copied from the design used in the ARToolKitPlus library.

5.2.1 Detecting BCH Markers

The detection of BCH code markers differs a lot from the detection pipeline implemented for the former marker layout. The only common part of the new detection pipeline (see Figure 5.3) consists of image acquisition, adaptive-threshold based binarization and morphological pre-processing. The following connected component analysis is already much simpler, since no region containment graph has to be computed. The connected component module can directly filter out regions by their size (pixel-count) and their value. By these means, white, too small and too large regions are filtered out. In the next step of the pipeline, the boundaries of the regions are traced. Unlike all other region features that are computed on the basis of the internal run-length-encoded (RLE) region representation, this is performed on the input binary image. The resulting list of contour pixels is internally memorized for later use. The number of boundary pixels and the region's pixel count are then used to filter out regions that are too elongated using a form-factor threshold (see Section 4.4.1).

From Region Boundaries to Quadrangles

In the next step, the region boundaries are approximated by polygons using a curvature scale space (CSS) based corner detection implementation [Mokhtarian and Suomela, 1998]. The CSS module has several parameters that were manually tuned for common conditions. Only the internal contour smoothing bandwidth is dynamically adapted with respect to the region size, allowing to more robustly detect corners of both very small and very large regions. For the resulting polygons (defined by the region corners), three cases are distinguished:



(a) thresholded image (b) intersection heuristic (c) perpendicular heuristic (d) mirror heuristic

Figure 5.4: **(a)** Bad thresholding parameters lead to a *broken* region with more than 4 corners. **(b)** The intersection heuristic allows a quadrangle to be reconstructed if two adjacent edges are broken. To this end, it extracts the longest edge *ab* and the second longest edge *ac*. The adjacent border segments *ce* and *bd* are extended yielding the intersection point *f* used as missing corner. **(c)** The perpendicular heuristic assumes an untilted view of the quadrangle, where all angles are square and all edges have a similar length. It rotates the longest edge *ab* by 90° around *b*, which yields *c*. Since even a minor perspective distortion of the quadrangle results in non-square angles, *c* is not used directly, but the closest original corner *d*. This step is repeated a second time, originating from *bd*. This provides a new guess *e*, yielding the last missing corner *f*, closest to *e*. **(d)** The mirror heuristic also uses the longest edge *ab* and the longest adjacent edge *ac*. The missing point *d* is obtained by mirroring the open corner *c* along the perpendicular bisector *X* of *ab*.

1. *The polygon has less than four corners.* In this case, no quadrangle can be extracted, so the corresponding region is dropped.

2. *The polygon has exactly four corners.* This is the most common case, where the quadrangle is directly passed to the corner optimization module.

3. *The polygon has more than four corners*. Usually, this implies that the regions does not correspond to a marker. However, in order to avoid dropping markers whose region borders have additional corners due to binarization artifacts, a set of heuristics is applied to extract potential sub-quadrangles from the polygon.

In contrast to the trivial cases of 1 and 2, case 3 involves three different heuristics that are applied to the initial set of region corners that are explained in Figure 5.4. The results of these are rated, allowing the likeliest solution to be selected. The use of heuristics for quadrangle reconstruction was also suggested and implemented by Fiala [2005]. However, the paper does not explain the used heuristics in detail and the corresponding software library is no longer available, so the heuristics were developed from scratch.

Quadrangle Rating

The rating, used to select the most consistent quadrangle reconstruction result, is based on a given quadrangle, defined by its four corners, $\mathbf{c}_i$, which are initially transformed to vectors $\mathbf{v}_i = \mathbf{c}_i - \mathbf{c}_{(i-1) \bmod 4}$. The rating is based on the five types of possible marker views presented in Figure 5.5. The rating $R \in [0, 1]$ is defined by a combination of sub-ratings:

$$R = \lambda(R_{\text{len}} R_{\text{angle}}) + (1 - \lambda) R_{\text{cross}}.$$

The length similarity rating R_{len} is defined by the relation of the average lengths of opposing edges:

$$R_{\text{len}} = \text{rel}(\|\mathbf{v}_0\| + \|\mathbf{v}_2\|, \|\mathbf{v}_1\| + \|\mathbf{v}_3\|),$$

(a) (b) (c) (d) (e)

Figure 5.5: Different types of marker perspectives **(a)** Untilted frontal view **(b)** diagonally tilted isometric perspective **(c)** diagonally tilted perspective projection **(d)** tilted isometric perspective **(e)** tilted perspective view.

where the relationship function $\mathrm{rel} : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow [0,1]$ is given by

$$\mathrm{rel}(a,b) = \min(\frac{a}{b}, \frac{b}{a}) \text{ and } \mathbb{R}^+ = \{x \in \mathbb{R} : x > 0\}.$$

R_{len} reaches its maximal value of 1 if the two pairs of opposing edges have the same average length ratio, which is true for Figure 5.5a-c. In addition to comparing the lengths of opposing edges, also the angles of opposing edges can assumed to be similar. Approximating the angle similarity of two vectors by their inner product, R_{angle} is defined by

$$R_{\text{angle}} = 0.5(\hat{\mathbf{v}}_0^\tau \hat{\mathbf{v}}_2 + \hat{\mathbf{v}}_1^\tau \hat{\mathbf{v}}_3),$$

which yields high values for untilted rectangles (see Figure 5.5a) or for isometrically projected rectangles (see Figure 5.5b,d). The multiplicative combination of R_{len} and R_{angle} yields good results most of the time. However, it prefers isometrically projected rectangles leading to issues with strong distortions caused by a perspective projection occurring when markers are close a camera with a small focal length (in particular in configurations similar to Figure 5.5e). This effect is compensated by the second additive sub-rating R_{cross}, which explicitly allows shorter opposing edges to be less parallel, while longer opposing edges may have a larger length difference. Let $q_{ab} = \mathrm{rel}(\|v_a\|, \|v_b\|)$, R_{cross} is given by

$$R_{\text{cross}} = \mathrm{rel}(\hat{\mathbf{v}}_0^\tau \hat{\mathbf{v}}_2, q_{13})\,\mathrm{rel}(\hat{\mathbf{v}}_1^\tau \hat{\mathbf{v}}_3, q_{02}).$$

A more sophisticated rating could be defined by extracting the 6D marker pose from a potential quadrangle. Assuming the original marker to be squared, the resulting re-projection error would yield a reliable consistency measure. However, this would assume known camera parameters entailing the necessity of performing camera calibration even for simple 2D marker detection. In addition, the presented rating always provides results $\in [0,1]$ allowing for the definition of a single scalar threshold to control the number of potentially reconstructed marker regions in an intuitive manner. The weighting parameter λ was manually tuned to a value of 0.5, which allows the system to reconstruct even very tilted quadrangles, while slightly preferring untilted ones[2].

Quadrangle Optimization

Once a quadrangle is extracted, the edges that were not created by one of the reconstruction heuristics are optimized by fitting straight lines into the set of boundary points between the edge start and end points. For a set of points $P = \{\mathbf{p}_i : i \in \{1,..,n\}\}$, this is performed by explicitly conducting principal component analysis (PCA). Let

$$C = \begin{bmatrix} c_{\text{xx}} & c_{\text{xy}} \\ c_{\text{xy}} & c_{\text{yy}} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{p}_i - \bar{\mathbf{p}})^\tau,$$

[2] The author thanks **Viktor Losing** for the ideas and the implementation of the heuristics and ratings

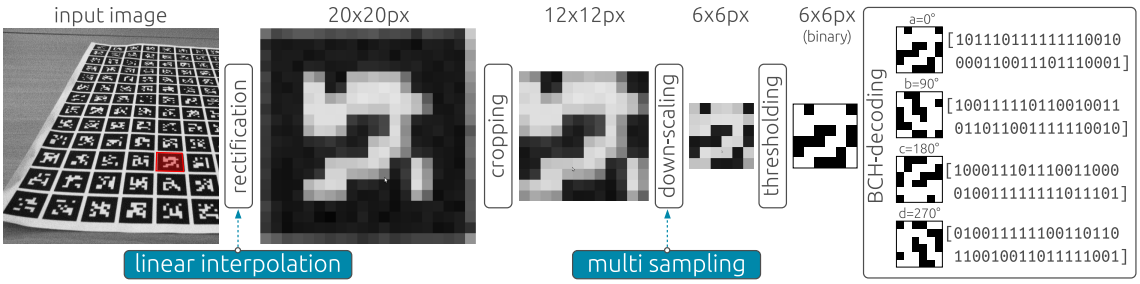


Figure 5.6: Marker identification pipeline. After rectifying the marker patch, the extracted image is cropped and down-scaled using multi-sampling. A following thresholding operation yields the BCH-binary code of the marker. Due to the fact that the marker rotation is unknown, all four possible rotations are evaluated. Given the properties of the used BCH-code, invalid rotations lead to invalid codes.

then the solution of the eigenvalue problem is given by $\det(C - \lambda\mathbf{1}) = 0$, yielding

$$\lambda_{1/2} = \frac{c_{xx} + c_{yy}}{2} \pm \sqrt{\frac{(c_{xx} - c_{yy})^2}{4} + c_{xy}^2}.$$

If $\lambda_1 > \lambda_2$, the eigenvector condition yields $\hat{\mathbf{v}}_1 = (c_{xy}, \lambda_1 - c_{xx})^\tau$ as a direction vector of the approximated line $\mathbf{l}(\alpha) = \bar{\mathbf{p}} + \alpha\hat{\mathbf{v}}_1$.
The adjacent optimized straight lines and the straight lines that can trivially be extracted from the heuristically created edges are then intersected to obtain optimized quadrangle corners. While this step provides only a moderate benefit for the 2D marker detection and the marker identification, it significantly improves the 6D pose estimation results.

Marker Identification

Each extracted quadrangle is used as a potential root-region for a marker. False positives are robustly removed by extracting the marker's BCH-encoded ID. The marker identification pipeline is depicted in Figure 5.6.
Due to the fact that the initial binarization step discards a lot of information, the following rectification step is performed on the original gray-scale image. 2D homography is used to rectify the marker center resulting in a perspectively undistorted 20×20 gray-scale image of a marker. The edge-length of 20 pixels is derived from the appearance of the markers. The marker design uses 2 size units for the marker border and 6×6 units for the binary BCH-code, resulting in a total of 10×10 units, so the rectified image is a, by a factor of two, scaled up version of the marker. Consequently the BCH-code of the marker can be extracted by simply cropping a four pixel border from the image, followed by downscaling the image by a factor of two. The BCH-code extraction is performed in two steps because during rectification, the image is only linearly interpolated, leading to interpolation artifacts in case of down-sampling. These artifacts are mostly removed by averaging every four neighboring pixels in a two step down-sampling approach. In order to obtain the binary BCH-code, an optimized threshold is estimated using a two-class vector quantization implementation. Let $P = \{p_1, .., p_{36}\}$ be the gray values of the pixels of the extracted marker BCH code, then the threshold θ is derived recursively starting with $\theta_0 = \mathbf{E}(P)$, where $\mathbf{E}(.)$ denotes the *expected value* [3]

$$\theta_i = \frac{1}{2}(\mathbf{E}(\{p_i : p_i \geq \theta_{i-1}\}) + \mathbf{E}(\{p_i : p_i < \theta_{i-1}\}))$$

[3] $\mathbf{E}(\{x_1, .., x_n\}) = \frac{1}{n}\sum_i x_i$

The mechanism converges quickly after a few iterations. Due to the fact that the marker rotation is unknown, the resulting 2D binary code must be decoded for each of the four possible rotations. In contrast to encoding information into a BCH code, which can be implemented by a simple binary multiplication of the word with a binary *generator polynomial*, the decoding step is much more complicated and is omitted from this thesis. The interested reader is directed to [Bose and Chaudhuri, 1960; Massey, 1969; Wang et al., 2001] for a detailed description of the process.

5.2.2 Paper Layout

The improved fiducial marker design allows more markers to be placed on the manipulated sheet of paper. Tests showed that a regular 9×13 grid of markers with an edge length of approximately 20mm printed on both sides of the paper provides a good trade-off between detection detail and accuracy. This led to a total of 234 markers, each providing its four corners as robustly detectable key-points on the paper surface.
While these numbers sound impressive in comparison to the 60 markers (5×6 per side) used in the picking-up experiment (see Section 4.2.1), there are some drawbacks. While each of the former markers provided an average of $8 - 12$ key-points, the new markers only provide four. This leads to a significant decrease of accuracy for the single-view 6D marker pose estimation. A detailed evaluation and comparison to the former fiducial marker types is presented in Section 5.4.

5.3 Modeling

The extended physical paper model implemented for the paper folding experiment, not only allows for the modeling of folds, but also for memorizing the deformation of the model. Both of these features are achieved by altering the parameters of the constraints of the physical paper model. Furthermore, our new model is set up with a new control mechanism that allows the tracking system not only to control the position of single nodes of the model grid, but also the position of arbitrary interpolated positions in between existing nodes.
Just like the original physical paper model (see Section 4.3.3), the extended model can be described by a regular grid of nodes $n_{\mathbf{p}^m}$, each associating the discrete 2D model coordinate $\mathbf{p}^m \in P' = \{1, .., W\} \times \{1, .., H\}$ of a node, with its current world location $\mathbf{x}^w_{\mathbf{p}^m}$. The resulting paper function $p : P' \rightarrow \mathbb{R}^3$ is again defined by the piecewise bi-linear interpolation of the model grid.
A constraint that connects two nodes $n_{\mathbf{p}^m}$ and $n_{\mathbf{q}^m}$ is denoted by $c_{\mathbf{pq}}$[4] and parametrized by its stiffness coefficient $s_{\mathbf{pq}}$ and its resting distance $r_{\mathbf{pq}}$. Constraints that connect adjacent nodes ($d_{\mathbf{pq}} = \|\mathbf{p}^m - \mathbf{q}^m\|_{L1} = 1$) limit the stretchability of the model and are not adapted. In contrast, constraints that connect not directly neighbored nodes ($d_{\mathbf{pq}} > 1$) limit the local curvature of the model and are therefore referenced as *bending constraints.*
In comparison to the original physics model, the set of used bending constraints was significantly reduced by dropping constraints whose referenced nodes have a model space city-block distance $d_{\mathbf{pq}}$ of 2. While qualitative tests showed that this has no noticeable effect on the modeling accuracy or stability, it significantly reduces the number of constraints used (depending on the dimensions of the physical model grid, usually by about 30%), allowing for a higher frame-rate of the physical simulation.

[4] the super script $.^m$ denoting that $\mathbf{p}$ and $\mathbf{q}$ are coordinates in the model space is omitted here for simplicity

5.3.1 Simulation of Folds

A fold is considered to be a straight line, l, in the model space. The parameters of bending constraints $c_{\mathbf{pq}}$ whose connection $\overline{\mathbf{pq}}$ intersects with l are adapted, allowing the modeling of two important aspects of folds. By significantly decreasing the stiffness coefficients of the intersecting constraints, the model becomes locally creasable resulting in a hinge-joint-like behavior of the model. At this point, the self-collision simulation of the physics-engine becomes important to avoid self-penetration of the paper model. However, hinge-joint-like behavior of real-paper can only be achieved by repeatedly folding the paper along the same line into opposite directions. When simply folded, paper usually shows plastic behavior resulting in memorization of external deformation.
For the simulation of plastic behavior, the resting distances $r_{\mathbf{pq}}$ of intersecting constraints also have to be adapted. Therefore, the modeling of folding behavior along a straight line l is performed in the following steps:

1. Find the set $C = \{c_{\mathbf{q}_i\mathbf{p}_i}\}$ of intersecting bending constraints
2. Lower the associated stiffness coefficients $\{s_{\mathbf{q}_i\mathbf{p}_i}\}$
3. Perform the folding operation along l
4. For each constraint $c_{\mathbf{q}_i\mathbf{p}_i}$, set the resting distance to the current distance of the referenced nodes in the world: $r_{\mathbf{q}_i\mathbf{p}_i} \leftarrow \|\mathbf{p}_i^w - \mathbf{q}_i^w\|$
5. Reset the associated stiffness coefficients to standard values

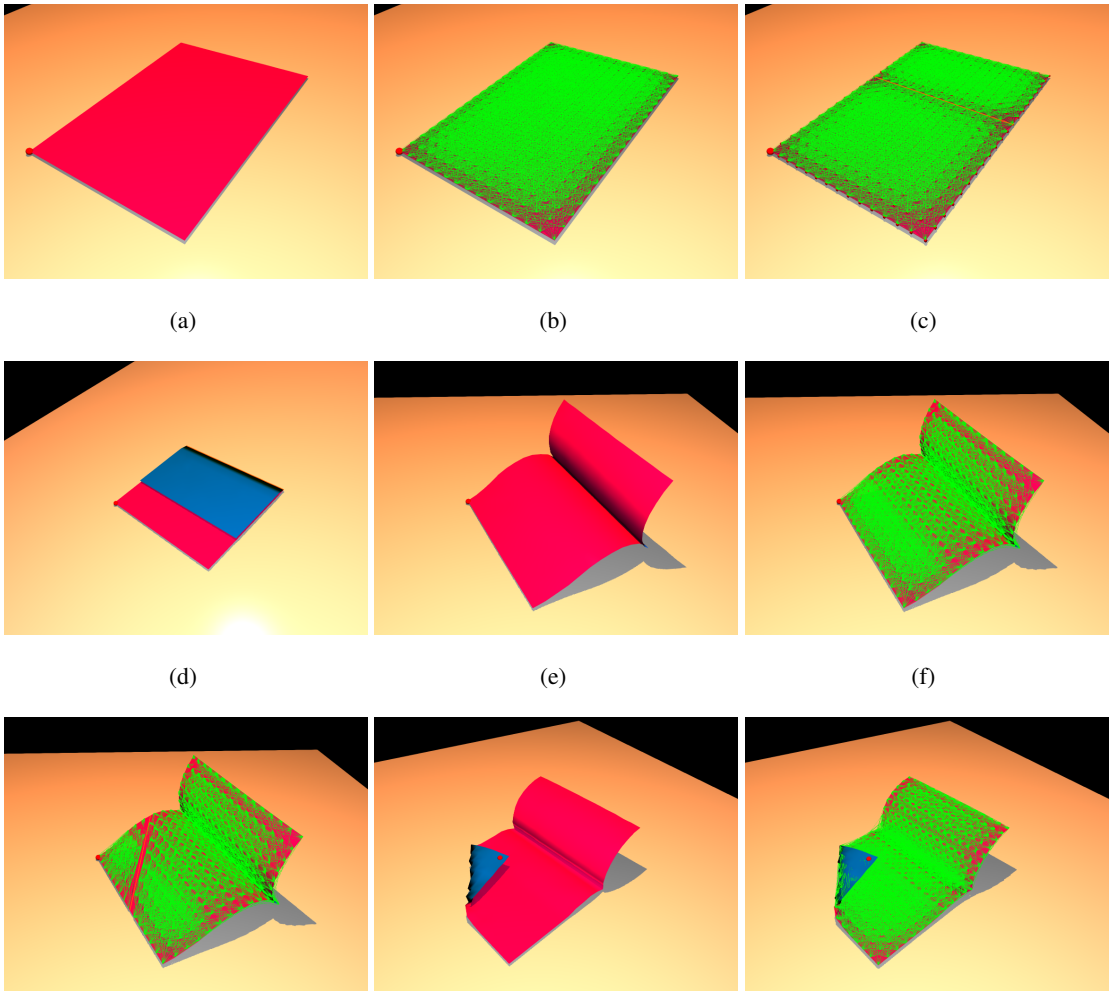
For the demonstration of this sequence, a GUI-based editor was created that allows mouse-gestures to be used to define fold lines and to move the paper in a drag-and-drop manner. The editor was also used for the demonstration of how fold lines are added and how they effect the set of bending constraints (see Figure 5.7). Furthermore, the editor allows random rigid boxes to be added to manipulate the global paper stiffness and to fixate nodes of the paper model manually. It is important to note that the achieved modeling of diagonal folds is sub-optimal since folds that are not aligned with the structure of the regular model grid are approximated by the model's grid structure. A more sophisticated model-adaption technique where fold lines are created by actually adding new paper-model nodes along the fold line is presented in Section 6.1.

Choice of Link Stiffness Values

The choice of scalar values for *normal* and *decreased* link stiffness strongly depend on the internal implementation of the bullet-physics engine, which expects the values to be in range $]0, 1]$. The values are not directly coupled to a physical quantity. Experiments showed that a global link stiffness of 0.9 results in a stable and paper-like model behavior. In contrast, lower values of 0.01 are well suited for modeling fold lines. Decreasing the global stiffness values makes the model behave more and more like cloth rather than paper (see Figure 5.8).

5.3.2 A Generalized Model Control Law

Due to the fact that model nodes and key-points are no longer required to be aligned in the 2D model space, a new control law was needed that allows for the adaption of the position of arbitrary model positions according to real world observations provided by the feature tracking component. While currently the corners and the center of the fiducial markers are used as features, the new control law paves the way for replacing marker-based features with more generic image features

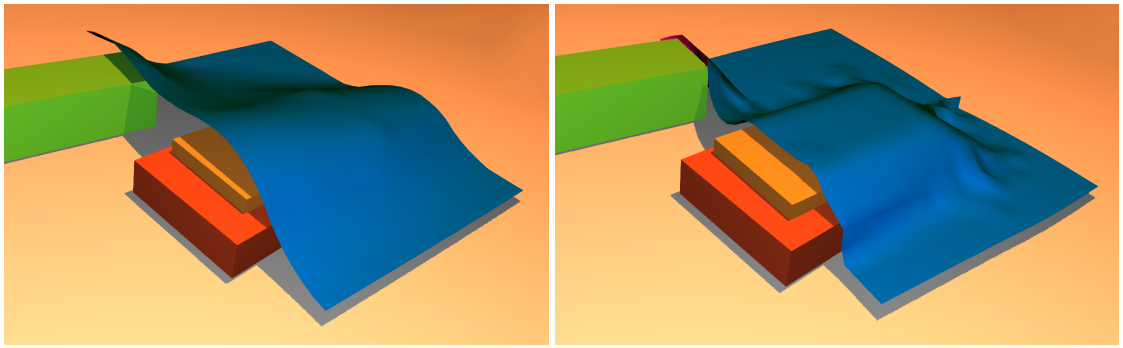


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.7: Simulation editor used for the creation and memorization of fold lines. **(a)** Initial paper model. The red sphere, initially located at the corner of the model, indicates the last point where the model was dragged by mouse. **(b)** Initial paper model, augmented with the full set of initial bending constraints. **(c)** A fold line was added along a grid line parallel to the short edge of the paper. All intersecting bending constraints were removed (internally, their stiffness coefficients are lowered to 0.01). **(d)** The paper model can now be folded along the fold line (using mouse drag and drop gestures). **(e,f)** After the memorization of the fold, the paper cannot simply be unfolded. The modeling engine tries to maintain the resting distances from the folded state. **(g)** An additional diagonal fold line was added, and again, all intersecting bending constraints were removed. **(h,i)** Now the paper is folded along the diagonal fold line. However, while the first fold parallel to the grid is completely straight, the diagonal fold appears slightly crinkled since it is approximated by the existing grid structure.



(a) high global stiffness (b) low global stiffness

Figure 5.8: Comparison of different values for the global link stiffness of the paper model. **(a)** uses a high global link stiffness of 0.9 leading to a stiff paper-like model behavior. **(b)** a low global stiffness value of 0.01 allows the model behave more like cloth. The gap between the paper model and the rigid box objects is a result of a relatively large collision margin that is necessary for the real-time rigid versus soft object collision simulation in the bullet physics engine.

computed from edge or texture information.
As in the original physical paper model presented in Chapter 4, the control law is independently applied for each observed key-point correspondence $(\mathbf{k}_l^m, \mathbf{k}_l^w)$. However, due to the missing direct association of the key-point k_l to a node of the physical model grid, the displacement of nodes now has to be distributed to the 4 nodes $n_{\mathbf{q}_i}$ closest to the key-point's model space coordinate $\mathbf{k}_l^m$. Since the model nodes are aligned in a regular grid, the 4 closest nodes are always given by the corners of the grid cell that contains the key-points model position $\mathbf{k}_l^m$. The new control law, illustrated in Figure 5.9, first computes the desired displacement

$$\mathbf{d} = \mathbf{k}_l^w - p(\mathbf{k}_l^m)$$

of model surface point referenced by the feature. Like before, the new control law works like a P-controller, acting iteratively on the node's velocity. In each iteration step, the velocity

$$\mathbf{v_i} = \alpha_i \lambda \mathbf{d} \tag{5.1}$$

is added to the velocity of each of the 4 closest nodes $n_{\mathbf{q}_i}$. The desired displacement of a specific node $n_{\mathbf{q}_i}$ is therefore not only weighted by the controller's proportional gain λ, but also by a node-specific weighting factor

$$\alpha_i = \max(1 - \|\mathbf{k}_l^m - \mathbf{n}_{\mathbf{q}_i}^m\|, 0) \tag{5.2}$$

that distributes the displacement relatively to the node's inverse distance to feature point in model space. Due to the fact that the used model space P' uses a unit length of 1 for the grid cells, a simple threshold that suppresses negative weights is sufficient here.
As a matter of fact, the new control law is a pure generalization of the original law. In cases where a feature point's model coordinate $\mathbf{k}_l^m$ is coincidentally identical to a model grid point $\mathbf{n}_\mathbf{p}^m$, the weighting α_i becomes 1 for the closest node $n_\mathbf{p}$ and 0 for the other 3 nodes, directly leading back to the original version of the control law presented in Section 4.3.3.

5.4 Evaluation

In this section, the extensions of the physical model and the capabilities of the adapted fiducial marker layout and the corresponding detection engine are evaluated. For the physical model, the

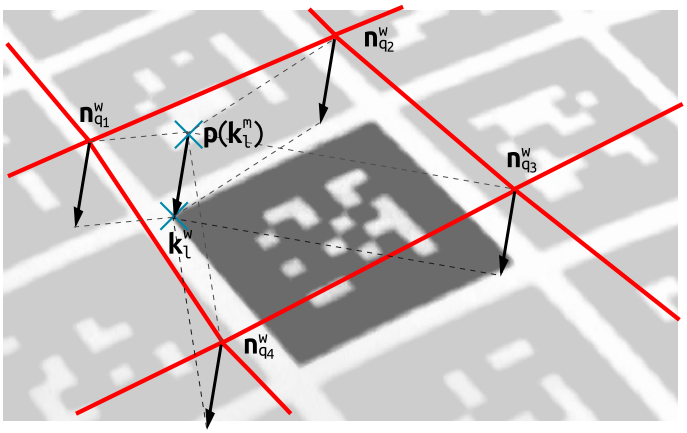


Figure 5.9: The current model is above the current observation of a marker. A key-point k_l, here corresponding to the marker's left corner (in image space), links an observed world position $\mathbf{k}^w_l$ to a position $\mathbf{k}^m_l$ in model space. The displacement vector of the current model position $\mathbf{p}(\mathbf{k}^m_l)$ of that point and the observed position $\mathbf{k}^w_l$ yields the base for the actual displacement of the 4 neighboring nodes $n_{\mathbf{q}_i}$. The displacement is weighted, dependent on the nodes distance to the key-point in model space.

ability to add fold lines and to memorize model deformation is put into focus. Since the evaluation provided in Section 4.4 conveyed an intuitive connection between qualitative results – usually presented by a model rendered as an image overlay, and quantitative values, this section will mainly focus on the presentation of qualitative results. However, due to the well defined scale given by the A4-sized paper, the presented images are also inherently linked to quantitative information. Due to the fact that the ability to model and to memorize folds offers a whole new level of manipulability, there is no direct comparison to the former version of the model, which did not allow folds to be modeled.
In contrast, the BCH markers are compared and contrasted with the previously used custom designed markers presented in Section 4.2. The comparison is performed quantitatively, on the basis of artificially rendered marker images, and also qualitatively as a part of the complete new paper-perception and modeling pipeline.

5.4.1 BCH-Code-based Markers

The new BCH-marker design is evaluated as an extension of the evaluation for the region-based markers, presented in Section 4.4.1. Again, the two aspects *identification* and *pose estimation* were evaluated separately. As the new BCH-markers can be detected from significantly further distances, the y-axes of the plots were adapted to cover the full possible distance range of up to 1.6m. As an extension of Figure 4.12, Figure 5.10 presents the robustness of the implemented detection system for BCH-markers. The figure shows that they outperform the region-based markers significantly. While the maximum detection distance of the region-based markers is about 600mm (frontal view) and 400mm (30° camera elevation angle), the BCH-markers are robustly detectable from up to double that distance. Furthermore, untilted BCH-markers (camera elevation angle about 90°) are still likely to be detected at more that 1500mm. The detection of BCH-markers is also much less prone to marker identification errors. In fact, the experiment revealed only two erroneous samples of none-crucial errors[5]. In comparison, the region-based markers are much more problematic because they are very prone to marker identification errors in the vicinity of their maximum detection distance.

[5] For these two samples, valid but unused marker IDs were computed.

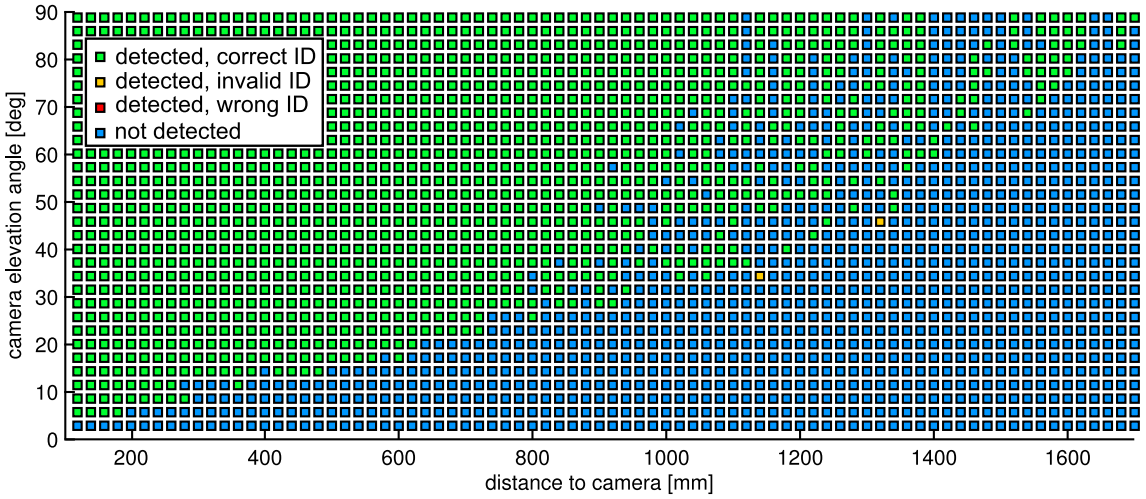
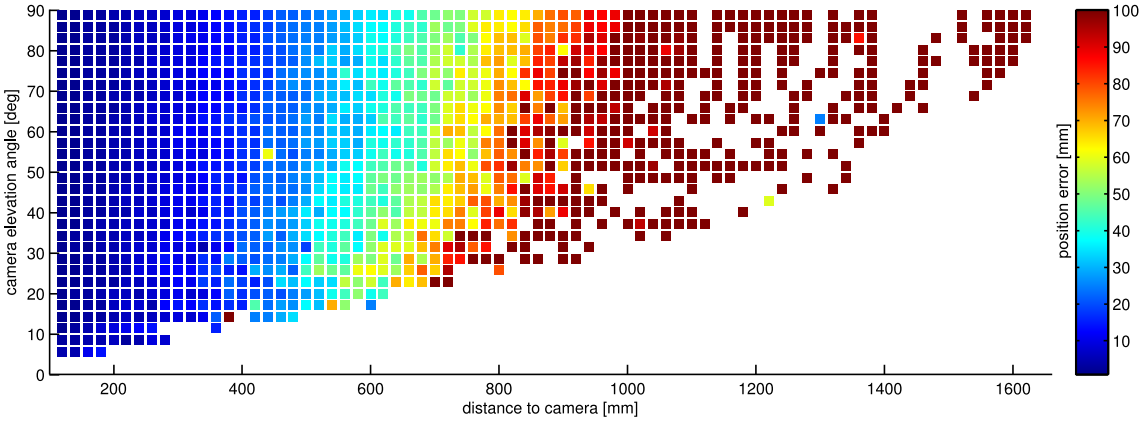
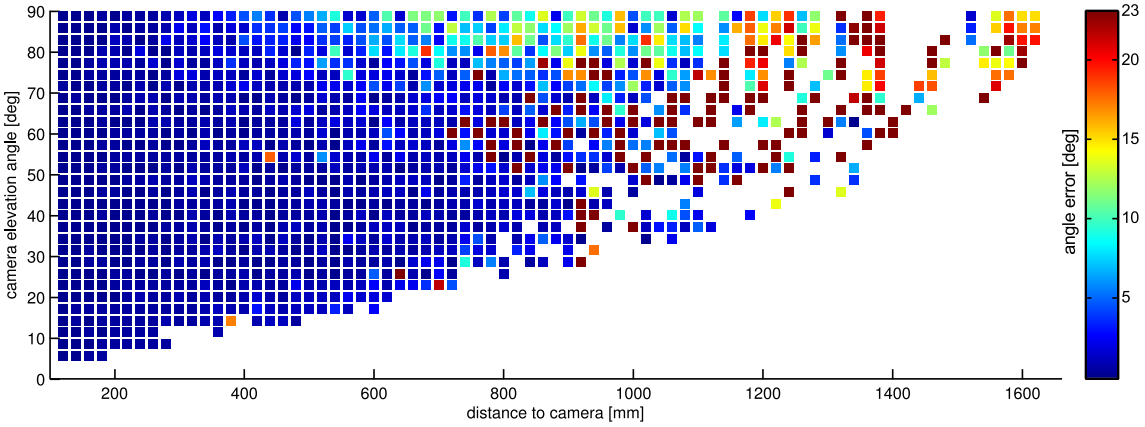


Figure 5.10: Evaluation of the robustness of the BCH-marker detection system, based on artificially rendered SVGA (800×600) images of square BCH-markers with an edge length of 30mm. For the experiment, 100 of the possible 4096 BCH-codes were picked and assumed to be the markers that are actually used. The two samples, where the ID estimation went wrong (yellow), yielded marker IDs that were not in this list, so the system would have been able to filter out these errors automatically.

With regard to the targeted tracking system, both aspects, detection distance and ID-estimation reliability are important. The higher possible detection distance provided by the BCH-markers indirectly allows a higher number of smaller markers to be placed on the tracked sheet of paper, yielding a denser grid of key-points. The increased marker identification reliability is, however, even more important for the system. Marker identification errors lead to erroneous key-points, which make the model control module move a part of the paper to an unrealistic world position. While the temporal physics-based model control law handles single instances of such outliers robustly, a larger or repetitive set of such errors would either lead to random model deformations, or even to numerical instabilities of the physics engine. These can usually only be detected after they have already occurred, so that an automatic avoidance or recovery is very hard.
Figure 5.11 plots the position and orientation estimation errors for BCH-markers. Even though the coarse shape of the error landscapes is comparable to the error plot of region-based markers (see Figure 4.13) , the overall pose estimation of BCH-code markers is significantly less accurate (see Figure 5.11a). While the average position error for the region-based markers in a representative distance of about 400mm to the cameras is only about 5mm, BCH-markers produce errors in the order of 20mm. The feature that BCH-markers can be detected from further distances even magnifies this issue here, leading to errors of more than 100mm. The low position estimation accuracy is mainly due to the thresholding operation that is used in the marker detection pipeline. The black-to-white transition of the marker boundaries lead, due to aliasing artifacts, to further gray pixels around the marker boundary. Dependent on the selected threshold, these gray pixels either become a part of the marker or not, resulting in radial errors in the order of one pixel for the boundary pixels, which are used to compute the corner key-points of the marker. When the marker is further away from the camera it appears smaller and the influence of this pixel-related error carries a higher weight in the pose-estimation.
In contrast, the estimation of the marker rotation (see Figure 5.11b) is more robust than for region-based markers, which is, however, due to the disadvantages of the position estimation of small use. While a robust estimation of the marker position could be used without reliable information about the marker orientation, an accurate marker orientation alone does not provide useful information for the tracking system. Therefore it was decided to employ more robust multi-camera



(a) position estimation error [mm](clipped to 100mm)



(b) orientation estimation error [deg](clipped to 23°)

Figure 5.11: Single-camera pose estimation error for synthetic SVGA input images of 30 × 30mm BCH-markers split into a position component (a) and a rotation component (b).

pose-detection methods only in the BCH-marker based tracking system.
In conclusion it can be stated that BCH-markers significantly improve the paper tracking system. The detection of the markers is fast and robust. The missing suitability for single-view marker pose estimation methods can mostly be compensated using a larger set of cameras to reduce the number of occluded markers.

5.4.2 Detecting and Modeling Paper with Creases

The capabilities of the new detection and modeling framework are presented using five different paper manipulation sequences, carried out by a human. The sequences are captured by a calibrated 6-camera setup described in Figure 5.12. The selected manipulation sequences are:

1. **Iterative folding** Here, the paper is folded in half 3 times iteratively in crossing directions. The system here has to deal with a simple fold and with folds that are applied to two or even four layers of paper simultaneously. Finished folds are memorized by the model.

2. **Paper aeroplane** A simple paper aeroplane is folded. This interaction sequence adds diagonal folds that have to be approximated by the model grid. In addition, the aeroplane folding sequence incorporates temporal folds that are not immediately memorized, leaving parts of the model in a hinge-joint-like configuration, not only while the corresponding fold

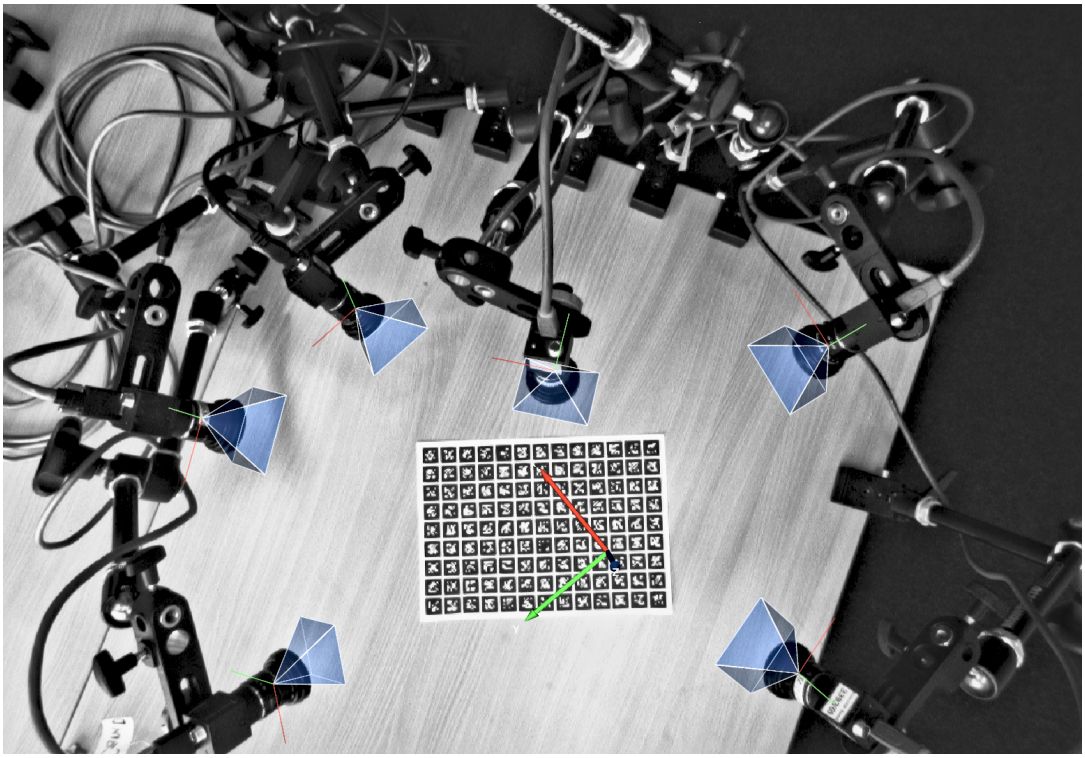


Figure 5.12: Calibrated 6-camera setup used for monitoring the human paper manipulation sequences. Each camera runs at 15Hz with quad-VGA resolution (1280×960). The whole setup runs on a single quad-core Intel ®Xeon ®E5530 (2.4 GHz) machine, endowed with three separate FireWire-800 buses. The 2D marker detection unit uses a dedicated thread for each camera, processing images at the video frame-rate. The 3D-estimation and modeling application concurrently receives the 2D fiducial marker detection results via shared-memory and performs 3D-point estimation and physical modeling asynchronously in an extra working thread, running at 5-40Hz (depending on the complexity of the self-collision state of the paper-model).

is applied, but also while the whole model is moved and while other fold lines are added to the model and their corresponding folds are made.

3. **Paper hat** A paper hat is folded. While the complexity of this interaction sequence is comparable to that of the aeroplane folding sequence, its final configuration consists partly of six closely overlapping layers, pushing the physical model's self-collision handling capabilities to their limit. In addition, some faces of the resulting paper hat are small enough, so that they do not even contain a single whole fiducial marker.

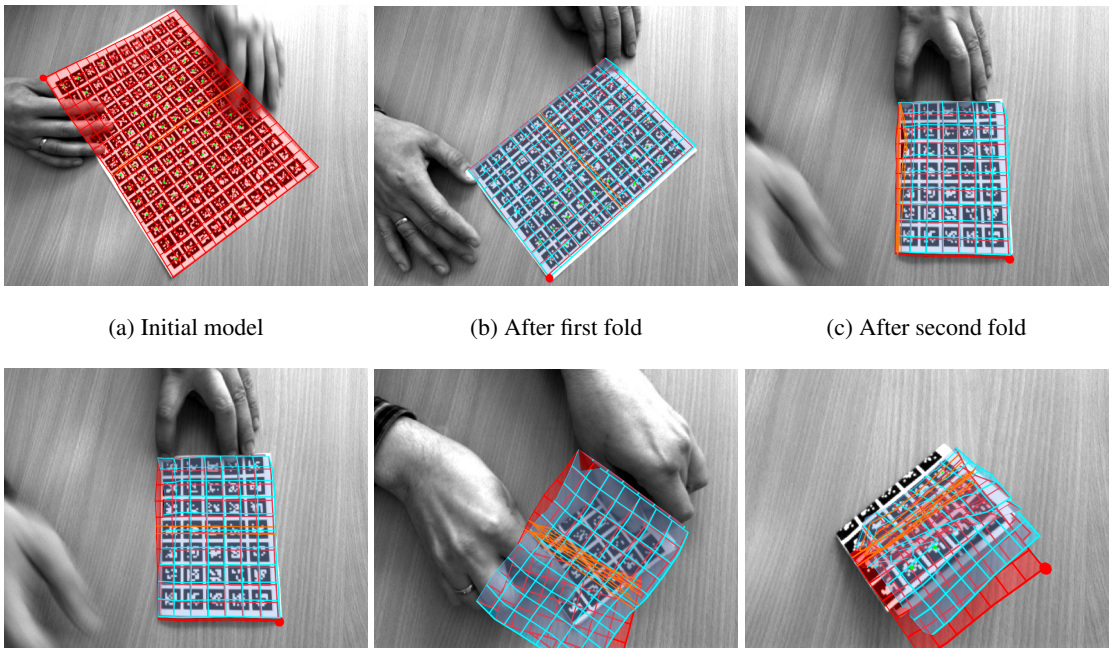
4. **Squash fold** A complex squash fold is performed, where two overlapping layers of the paper are squashed from the top to produce an *inner bag*. Here, the system has to deal with extreme occlusions, caused by the manipulating hands as well as by the overlapping layers of the paper. Additionally, the very complex physical deformation of the paper should be reflected by the model.

5. **Crushing** The whole sheet of paper is crushed, first slightly and then more heavily. This interaction sequence differs most from the other examples as no distinct fold lines can be used. Instead, the global model stiffness is decreased in order to allow the model to reflect the desired deformation.

During the interaction sequences, the lack of an automatic fold line detection, was compensated by manually adding fold lines to the model using mouse drag and drop gestures in the model editor,

which is integrated into the 3D estimation and modeling application. From here, the memorization of the current deformation of the paper can also be triggered manually on demand.

Iterative Folding



(a) Initial model (b) After first fold (c) After second fold

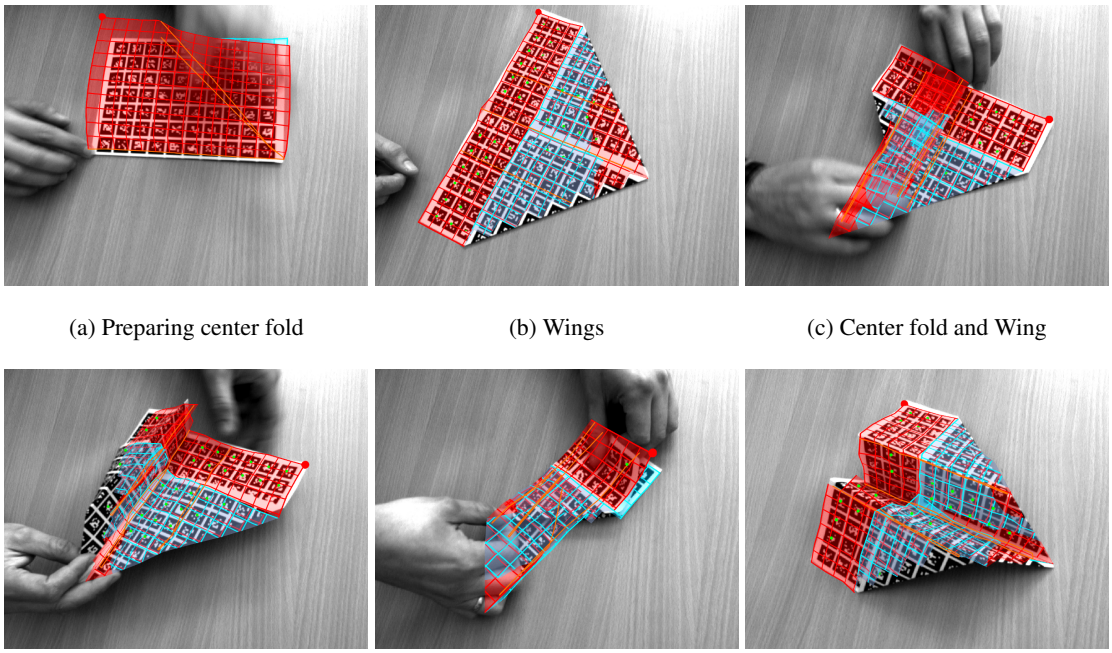
(d) Before third fold (e) During third fold (f) After third fold

Figure 5.13: Iterative folding manipulation sequence. While two iterative folds can be tackled well by the system (a-d), the third fold (e,f) leads to major differences between model and observation.

As a first step, the system was used to track a sheet of paper that is iteratively folded three times in cross-wise directions. Before the folds are performed by the human, the next fold line is added manually to the model, leading to a hinge-joint like model behavior along that line. This means that the fold line is added to one, two or four overlapping layers of paper respectively. After a fold is performed, the folded state of the model is memorized to avoid an *accidental unfolding* while the following folds are carried out. Figure 5.13 visualizes the results of the experiment. The results show that only two iterative folds can be tracked well by the system. There are two major reasons that explain this. First, the system has to deal with extreme occlusions (see Figure 5.13e), leaving only a very few markers visible while the last fold is performed. Second, the fact that folds are only approximately added manually to the model leads to a set of error sources. The used mouse-input method weakens all constraints that intersect with the 3D-plane defined by the mouse drag and drop gesture. The plane is derived from the three points defined by the camera origin and the intersections of the mouse drag and drop point's view rays with the z=0 plane. While this allows for an intuitive manual definition of fold lines, it also entails that fold lines in overlapping layers of the paper are only perfectly aligned in the image space of the manipulating camera view, but not necessarily in the model space. This effect can be compensated to a certain degree by adding two adjacent parallel fold lines, in order to create a *broader* fold line (see Figure 5.13b,d). However, the presented example shows the limitation of this fix. A more precise hand-crafted programmatic creation of fold lines would solve this issue, albeit leading to a loss of generality of the system.
Another issue that occurred when performing iterative folds concerns the memorization of deformation, which is implemented by simply reading out the current model configuration to memorize

the current distance of each two nodes referenced by a bending constraint as its new resting distance. Due to the internal self-collision handing of the physical model that employs a *collision margin* to stabilize the internal system, folds are never precisely $180°$ and two folded layers will never really touch. Therefore, the collision margin basically defines the modeled thickness of the sheet of paper. However, due to numerical reasons, the minimum thickness that can be modeled properly turned out to be in the order of 3mm, i.e. about 30 times thicker than real paper. Due to this issue, each memorized fold of two overlapping layers leads to *tension* in the model. This tension is even amplified in case of four or more overlapping layers. To overcome this effect, an explicit handling of folds would have to be added to the modeling engine. Once a fold is performed and is to be memorized, the rectangular model would have to be replaced by a polygonal one that internally models the folded two layers of paper as a single layer. However, this approach would not allow a fold to be undone and leads to several further non-trivial issues, both theoretical and practical in nature.

Folding a Paper Aeroplane



(a) Preparing center fold (b) Wings (c) Center fold and Wing

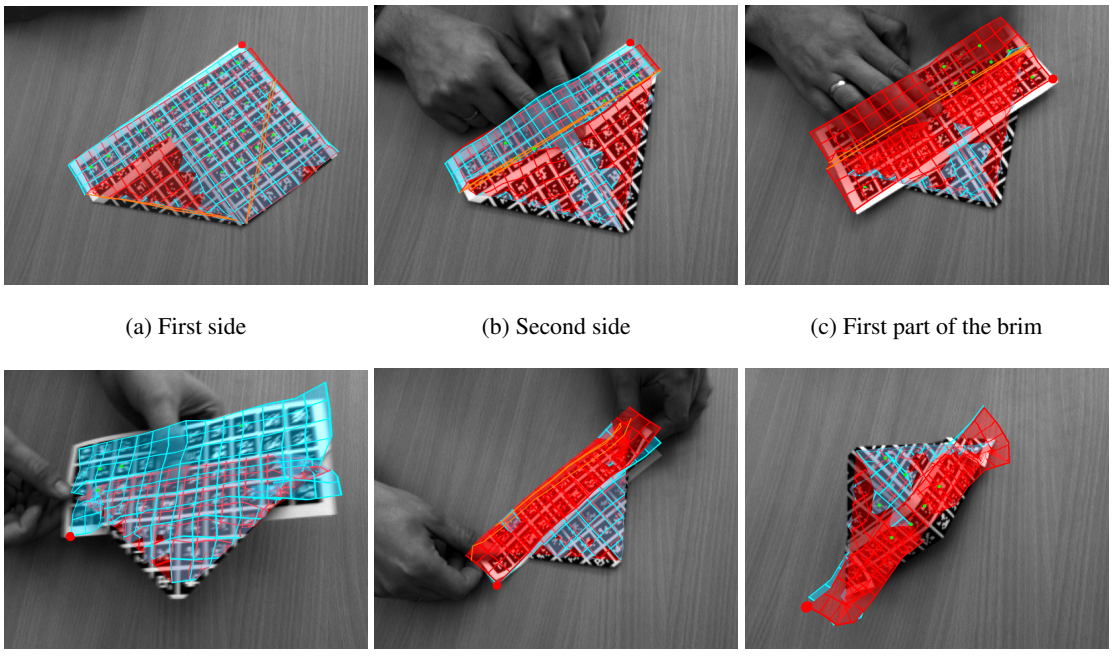
(d) Turn around (e) Abduct left wing (f) Final paper aeroplane

Figure 5.14: Tracking a sheet of paper while it is folded into a paper aeroplane. The detection and tracking system performs very well during the whole interaction sequence. Even temporary and diagonal folds can be successfully modeled and tracked.

The iterative folding sequence covers only a small set of common paper folding aspects. In particular, it does not employ diagonal folds or steps in which folds are inverted. Therefore, folding a paper aeroplane seemed to be a reasonable next step[6]. Furthermore, a paper aeroplane is one of the first complex objects many children learn to fold. The steps of the manipulation sequence are shown in Figure 5.14. The center fold line and the diagonal fold lines for the wings are initially added to the model. Before the wings can be folded diagonally (see Figure 5.14b), the center fold line of the real paper is created by temporarily folding the paper in half (see Figure 5.14a). After memorizing the diagonal wing folds, the fuselage of the aeroplane is created. Here, the whole

[6] The tracking of the paper aeroplane folding sequence is also shown in the video [Elbrechter et al., 2012b].

partly folded paper aeroplane is turned around (see Figure 5.14d) revealing that tracking of a complex deformed model is well supported by the system.
The most problematic part of this manipulation sequence is the necessity to precisely define the fold lines of the model. While fold lines like the one for the center fold can easily be defined, the exact definition of the diagonal fold lines is not trivial. Actually also fold lines parallel to the model grid can easily become problematic in cases where the targeted fold line is not close enough to a model grid coordinate. As soon as the model fold lines and the folds of the real paper differ more then minimally, the tracking performance significantly decreases. A crucial parameter, inherently linked to this issue is not only the key-point density, i.e., the density of the fiducial marker grid, but also the resolution of the physical soft-body grid of nodes. The needed key-point density depends on the smallest face of the created model. For the used marker-based key-points, faces must be large enough to show at least one marker completely. Otherwise, the face provides no external information to the tracking, leaving the positioning of the face over time to the constraints of the physical model.
In contrast, the dependency on the resolution of the physical model leads to several interesting aspects. While in general, a higher model resolution results in a more flexible model and therefore increases the likelihood that an actual, in particular diagonal, fold can be exactly reflected by a model fold line, it significantly lowers the real-time performance of the system. In turn, since the model is tracked over time, this also affects the resulting spatial step-width between two successive effectively used frames, resulting in a higher possibility to *loose* the model.
The uncomfortable trade-off between model flexibility and tracking-speed could be compensated by precisely modeling arbitrary fold lines by inserting new nodes along a to-be-inserted fold line. This mechanism was implemented for the model presented in Section 6.1.



(a) First side (b) Second side (c) First part of the brim

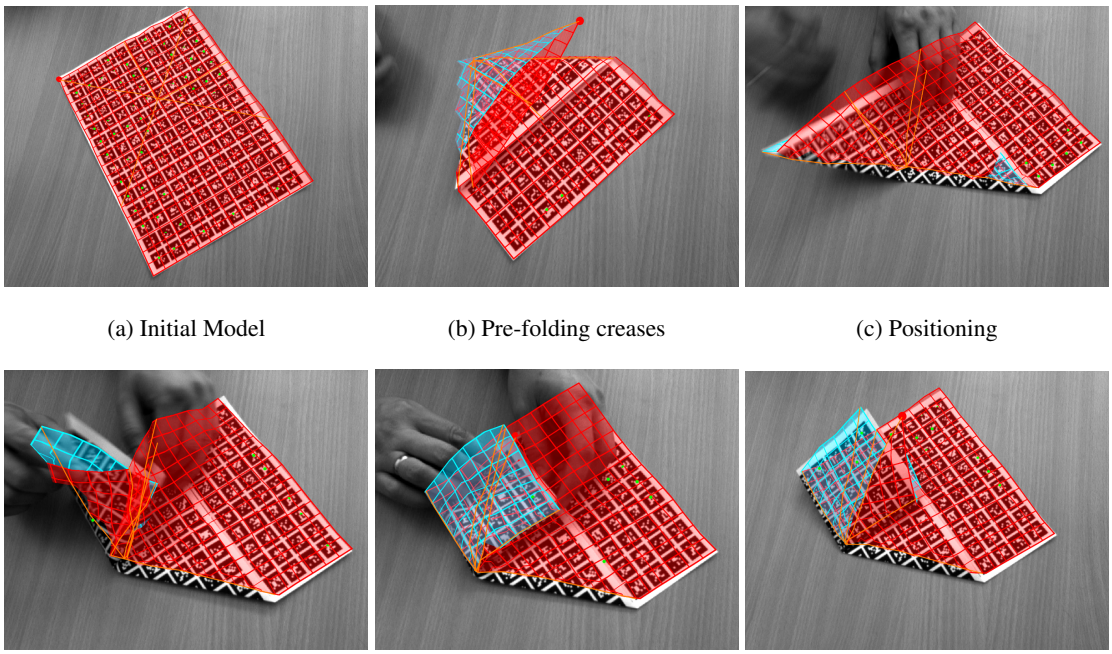
(d) Turning around (e) Second part of the brim (f) Final paper hat

Figure 5.15: Observing the deformation of a sheet of paper while it is folded into a paper hat by a human. The coarse tracking performance is satisfying, but the small triangles that fixate the brim of the hat (f) cannot be tracked since their corresponding paper face covers only a part of a fiducial marker.

Folding a Paper Hat

The hat-folding experiment (see Figure 5.15) shows that even six layers of paper can be modeled and tracked satisfyingly. However it must be admitted that the interaction was explicitly performed slowly in order to compensate the low tracking frame-rate of down to 5Hz as soon as too many layers of the paper overlap (see Figure 5.15c-f). Additionally the sharp corners of the hat's brim demonstrate the limitations of the system, because the corresponding small triangular paper faces only contain a fraction of a single fiducial marker. Since there is no physical constraint that could *draw* these triangles around the brim, they cannot be tracked properly. A manual adaption of the model could be used to fix this issue. The fact that the diagonal fold lines of the model do not perfectly coincide with the actual folds of the paper does not negatively affect the tracking performance.

Performing a Squash-Fold



(a) Initial Model (b) Pre-folding creases (c) Positioning

(d) Opening inner bag (e) Squashing (f) Final model

Figure 5.16: Observing the deformation of a sheet of paper while a human performs a squash fold.

As motivated by Tanaka et al. [2007], a squash-fold, where a double-layered part of the paper is *squashed* in order to produce an inner bag, is significantly more complex then other types of folds. Squash folds are even used as examples of complex folds by Balkcom and Mason [2008]. The increased complexity of squash folds not only makes it more difficult to design robots to perform them, but also means that the vision and modeling system is severely tested. The visual tracking of a squash fold is particularly difficult because squashing the paper leads to severe occlusions that need to be compensated and this requires an appropriate model, able to reflect the principle of a squash fold. Figure 5.16 shows the performance of our system at various key frames as a human performs a squash fold.
The results show that the system almost perfectly tracks the squash folding manipulation (see Figure 5.16c-e). However the final state (see Figure 5.16f) reveals a problem of the modeling

engine. Since the deformation of the paper is not explicitly memorized, a glitch in the model's self-collision handling allows the inner squashed part of the paper to unrealistically penetrate the top layer, resulting in a wrong final model state. Memorizing the squashed state of the model before the error occurs, would help to minimize this effect. The actual issue, however, seems to derive from numerical instabilities in the Bullet physics engine's collision handling and it is not unreasonable to expect that this will be improved upon in a future release.

Crushing Paper

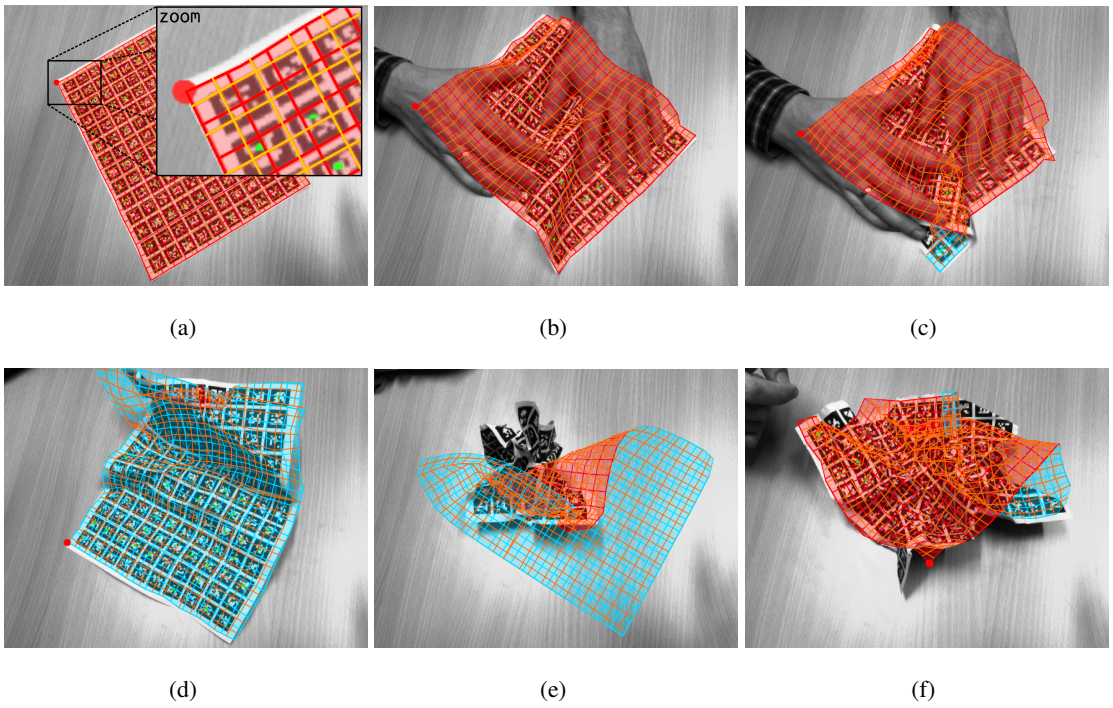
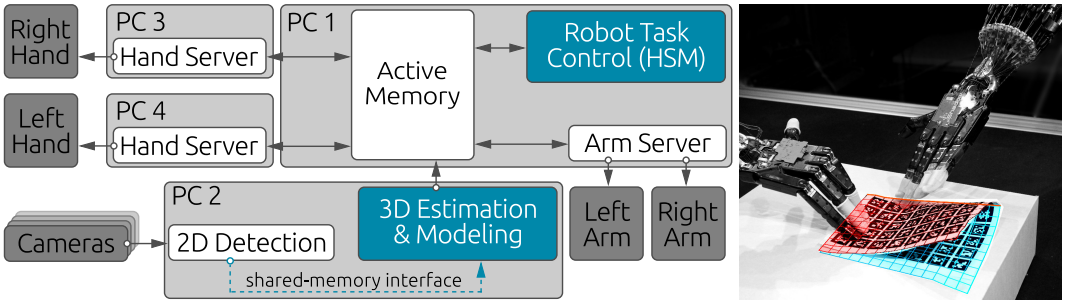


Figure 5.17: Observing the deformation of a manually crushed sheet of paper. **(a)** The model is made fully flexible by adding fold lines (yellow/orange) through every cell of the regular model grid (red). **(b-d)** Average deformation is satisfactorily tracked by the model. **(e)** Stronger deformations lead to major differences between observation and the model. **(f)** After partly undoing the deformation, the qualitative impression becomes acceptable again.

In contrast to the well defined folds and fold lines used for the other manipulation sequences, crushing leads to more general deformation of the paper. To globally reduced the model's stiffness, it is initially made fully flexible by manually adding a grid of fold lines so that every model cell is covered (see Figure 5.17a). This step is comparable to adapting the model's global stiffness resulting in a more *cloth-like* behavior (see Section 5.3.1). In the case of minor deformations (see Figure 5.17b-d), most of the fiducial markers remain detectable leading to robust tracking results. However, due to the low paper stiffness, invisible parts (e.g. in Figure 5.17c, the parts hidden by the human's hands) are unrealistically smoothed. Since the model is only loosely constrained, a possible solution would be to physically model the manipulating hands.
As soon as the average curvature of the sheet of paper increases to the point that most markers become undetectable, the tracking accuracy decreases dramatically. While the general position and rough orientation of the model is still tracked as long as a single whole fiducial marker remains visible to the system, the tracking of the deformation fails completely. If the totally crushed sheet of paper is slightly straightened again (see Figure 5.17f), the tracking performance of the



(a) (b)

Figure 5.18: **(a)** Data-flow and inter-process communication (IPC) scheme used for the paper folding experiment. IPC is implemented by a combination of the XCF/Active Memory infrastructure and shared-memory based communication. A single *2D Detection* module processes all camera images in dedicated threads and publishes the detection results using the shared memory interface. The *3D Estimation & Modeling* unit combines these results and inserts the current 3D paper model into the single *Active Memory* instance. The *Robot Task Control (HSM)* also uses this *Active Memory* instance to communicate with the robot servers. **(b)** Adaption to the Robot workspace. The folding manipulation was carried out on a raised support box. The use of a cardboard box increased the height of the table-top, which optimized the action radius of the robot hands and provided an *edge* that could be used by the robot to grasp a corner of the paper. In addition, the flexibility of the cardboard box lowered the risk of damaging the hand during the interaction sequence.

deformation can recover.

5.5 Robot Control

Using the new paper tracking and modeling framework, a robot interaction sequence was implemented that enabled our bi-manual anthropomorphic robot to fold a sheet of paper in half. Obviously, this manipulation does not fully exhaust the capabilities of our tracking and modeling framework. However, its accurate real-time feedback even in the presence of strong occlusions caused by the robot hands allowed us to fully focus on the unsolved robot-control related issues. The first task is to figure out how the folding sequence, sketched in Figure 5.1, can be transferred to the robot in general. In particular, it is not initially clear how the paper needs to be placed with respect to robot, and how well the accruing interaction primitives, such as fixating, pinch-grasping and creasing, can be implemented on the robot. As a sub-topic of this, we examined in which way tactile feedback can or must be called upon to supplement the feedback provided by the vision-framework. Furthermore, we explored, how the system can be made more robust with respect to changing starting conditions, such as the initial paper position and orientation.

5.5.1 Updated Vision and Robot Setup

Originating from the software setup used for the picking up experiment (see Section 4.5.1), the whole system was significantly simplified and optimized (see Figure 5.18a). As was shown in the evaluation (see Section 5.4.2), the improved vision and modeling framework can be run on a single PC, which not only reduces the complexity of the whole software setup, but also allows for further performance optimizations. In particular, this allows the original XCF/Active Memory-based conflation of the 2D fiducial marker detection results to be replaced by a shared memory

interface. This reduces not only the data transfer latency, but also significantly lowers the computational overhead arising from otherwise necessary massive XML parsing performed by the Active Memory server.
In comparison to the picking up experiment, the robot workspace was adapted by using a cardboard box as a raised support plane for the manipulation task (see Figure 5.18b). The edge of the box simulates a table edge that is, as motivated in the introduction of Chapter 5, often used by people to simplify the grasping of a corner of a sheet of paper. Furthermore, the support-box raises the level of the table plane, which means the robot forearms can be oriented more parallel to the table, increasing the robot's interaction radius. As a further advantage the box significantly lowered the risk of damaging the robot during implementing and testing of the folding sequence.
Our robot hands are equipped with different types of tactile sensors (see Figure 2.3a). The right hand's fingertips are equipped with a tactile sensor matrix comprising 34 taxels[7], providing a spatial resolution of 3mm, but at the expense of a rather limited sensitivity. On the left hand we employ PST sensors from the Shadow Robot Company that have no spatial resolution but have a very high sensitivity over a range of small forces. Therefore, we make asymmetric use of the hands and employ the right hand for fixating the paper and the left hand for the manipulation actions. To this end high friction rubber finger covers were added to the right hand's finger tips to facilitate fixating, while the friction of the left hand's finger tips was decreased by endowing them with fabric covers.

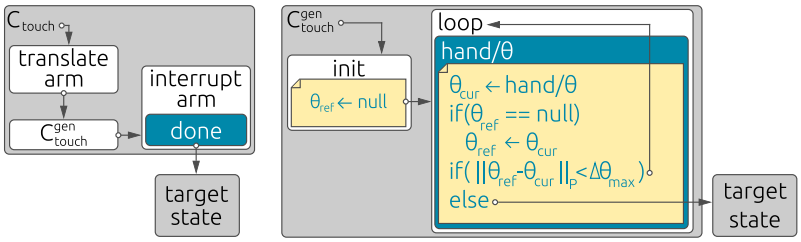
5.5.2 Registration of Reference Objects on the Robot Server

In order to simplify robotic object manipulation an updated version of the robot arm server was employed that provides a mechanism allowing objects to be manually registered on the server. The registered objects can be updated externally and the robot control primitives can be formulated relatively to the coordinate frame of a registered object. The extended syntax can handle different coordinate frames for the rotation and for the translation part of a movement command and it allows for the definition of a translation offset. By combining different frame assignments, absolute, as well as relative, movements can be performed with respect to an arbitrary coordinate frame. By internally performing all the necessary coordinate frame transformations between the object frame, the world frame and the robot end-effector frame, this feature allows for a significant reduction of both the complexity and the length of the robot controller code.
In the robot folding sequence, a single *paper* object was registered. However, since the object registration mechanism can handle rigid objects whose coordinate frames are defined by single rigid homogeneous transforms only, the paper frame was defined to be located at the center of the sheet of paper. As the paper is bent, only fiducial markers that are coplanar with the box's top surface are used to compute the paper coordinate frame.

5.5.3 Closed Loop Feedback Controllers

The previous systems for shifting (see Section 2.2) and for picking up paper (see Chapter 4) already employed closed loop controllers, but these were implemented explicitly in the robot control HSM. Due to the loose coupling (via network and XCF/ActiveMemory) between HSM and the robot servers (see Section 2.1) and because of the explicitly implemented perceive/action loops, only slow feedback cycles could be achieved, which resulted in very slow and jerky robot move-

[7] Tactile pixel



(a) (b)

Figure 5.19: Two-stage implementation of the feedback based controllers C_{touch} and $C^{\text{gen}}_{\text{touch}}$. **(a)** The C_{touch} controller is implemented as a reusable sub-HSM and it internally employs $C^{\text{gen}}_{\text{touch}}$ to allow an object to be touched. **(b)** The generic $C^{\text{gen}}_{\text{touch}}$ controller is implemented using python code (here explained using pseudo-code) embedded into the HSM description.

ments[8]. To improve this, both the hand and the arm server were extended by built-in interfaces[9] for the implementation of closed loop controllers.
While sketching the robotic folding sequence, two repeatedly occurring sub-patterns were identified, which led to the development of two different types of controllers, C_{touch} and C_{force}. Analogous to the explicitly implemented perceive/action loop for contact establishment in the picking up system (see Section 4.5.2), C_{touch} allows the robot to establish contact with the paper by exploiting the robot hand's passive compliance to estimate the contact force on the basis of the hand-posture displacement induced by an arm movement. In contrast, C_{force} allows a given contact force, caused by the finger-tips touching the paper, to be maintained.

Feedback-based Contact Establishment

The first closed loop controller that we implemented was C_{touch}, which was designed for feedback based contact establishment between the robot hand and an object. The actual implementation is split into a generic component, $C^{\text{gen}}_{\text{touch}}$ (see Figure 5.19b), and a more specific includable *convenience*-wrapper HSM, C_{touch} (see Figure 5.19a), that employs $C^{\text{gen}}_{\text{touch}}$ in order to make the robot touch an object (see Figure 5.19).
Inspired by the method successfully employed to establish contact with the paper in the picking-up experiment (see Section 4.5.2), $C^{\text{gen}}_{\text{touch}}$ initially registers the robot hand's current joint angles as reference posture, θ_{ref}. Alternatively, a given posture could be used as reference, but tests revealed that the desired hand posture and the actually actuated hand posture often differed too much. Once θ_{ref} is known, an arm-motion is performed until the current hand posture θ_{cur} differs by more than a specified threshold, $\Delta\theta_{\text{max}}$, from the reference posture:

$$\text{contact established if } \|\theta_{\text{cur}} - \theta_{\text{ref}}\|_P > \Delta\theta_{\text{max}},$$

where $\|.\|_P$ is the normalized Euclidean distance with diagonal weighting matrix P and $\Delta\theta_{\text{max}}$ is the posture difference threshold. Usually, P contains diagonal entries $\in \{0, 1\}$, allowing us to manually select hand joint angles that are to be taken into account for the stop criterion. Subsequently $C^{\text{gen}}_{\text{touch}}$ triggers a transition into a selectable target state. As $C^{\text{gen}}_{\text{touch}}$ does not specify the robot actions that actually induce the change of the hand posture, it can be seen as a concurrent interrupt mechanism.
In order to reduce the amount of boiler-plate code that was necessary to employ $C^{\text{gen}}_{\text{touch}}$ to actually

[8] A single perceive/action cycle consisting of a downwards movement and a subsequent sensor-readout could easily take some seconds.

[9] The author thanks Robert Haschke for his help with the ideas and the implementation of the interface and the controllers

touch the paper, the more specialized C_{touch} controller was implemented (see Figure 5.19a) that internally employs $C^{\text{gen}}_{\text{touch}}$. C_{touch} initiates an arm movement before entering $C^{\text{gen}}_{\text{touch}}$. It is important that this movement actually (after some time) leads to the desired change of the hand posture, which causes C_{touch} to immediately interrupt the arm movement before exiting to a selectable target state.
Due to the passive compliance of the air-pressure controlled robot hands, joint deviations can directly be translated to applied forces according to a nonlinear spring law. Therefore, not only the posture difference threshold, but also the joint stiffness employed for the initial hand posture θ_{init} can be used to select an appropriate contact force.
Alternatively, sufficiently sensitive tactile feedback could be used to estimate contact forces. However, even though a variant of the $C^{\text{gen}}_{\text{touch}}$ controller employing tactile feedback was initially implemented, this feedback channel was subsequently not used. In addition to the unfortunate placing of the touch sensors (see Section 2.2), the main reason for this was that the measured tactile feedback based contact forces were very sensitive to changes in the angle between the touched surface normal and the sensor orientation. Here, the passive hand compliance is actually disadvantageous as even a stiffly actuated contact hand posture allows the robot fingers to comply with the surface, which alters the angle between sensor and the touched surface and thus makes it very hard to select an appropriate force threshold.

Actively Maintaining Contact Force using Tactile Feedback

The second feedback based controller that was implemented for the paper folding experiment was used to maintain a contact force between the finger tips and an object surface. In contrast to the C_{touch} controller that was implemented as an includable HSM-substate, the C_{force} controller was implemented as a native feature of the hand server. The basic feature of the hand server process is to actuate hand postures defined by a posture vector and a joint mask. In the standard version of the hand server, the hand controller would simply try to maintain the target posture. An optional stiffness parameter defines the maximum air pressure value that is internally used to counteract contact forces with workspace objects.
The extension of the hand server features so called *active postures*, which are enriched with target tactile sensor values for selected finger tips. When receiving an active posture, the hand server internally spawns a controller process that actively maintains the target tactile values. Since the robot hand's finger-tip sensors are sensitive on the inner side only, closing the hand increases the contact force and opening the hand decreases it. Due to this direct relationship, a simple P-controller driving the finger flexion yields good results:

$$\Delta\theta_i^{\text{flex}} = \alpha_i \cdot k_p \cdot (f_{\text{target}} - f_{\text{current}}).$$

The α_i define joint-specific gain modulation factors that are needed to accomplish stronger flexion of proximal joints. Otherwise fingers would tend to curl inwards, which would, in turn, result in loosing contact with the tactile-sensitive palmar surfaces of the fingertips.
As the C_{force} controller only acts on the hand-joints, it can easily be combined with arm movements tangential to the touched surface. By implementing the controller as a native part of the hand server, it is not only reusable in a much simpler manner, but also significantly reduces the network communication load of systems that deploy it.
For the folding experiment, two different active hand postures were used. The first active posture employs all fingers except for the thumb, which is not used as in the given hand-arm-paper configuration its kinematics do not allow its tactile sensors to be sufficiently oriented towards the support box surface. This posture is used for shifting the paper and for swiping over the folded

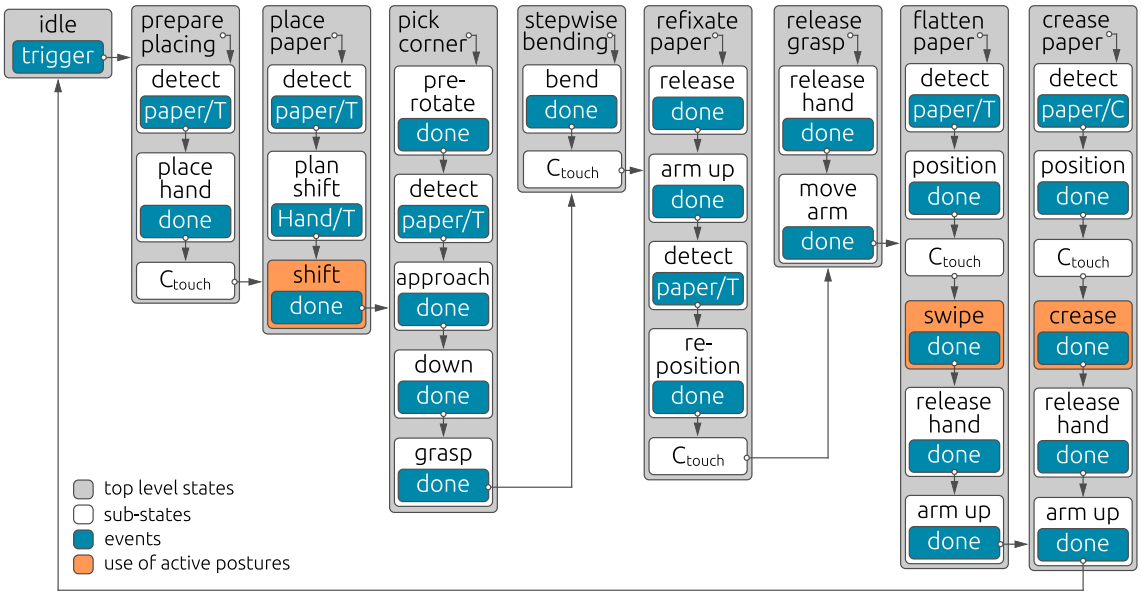


Figure 5.20: Hierarchical state machine (HSM) used for robot control in the folding paper experiment. The whole structure is implemented in a linear fashion. The remaining feedback loops are either encapsulated within the C_{touch} controllers or natively implemented as active postures (orange sub-states) using C_{force}.

paper to prepare the crease. The second active posture employs only the index and the middle finger and is used for a final precise hardening of the fold.

5.5.4 Folding Paper With the Robot

Once the feedback-based controller C_{touch} and the active posture controller C_{force} were available, the final robot control system for bi-manual robotic paper folding was implemented[10]. Similar to the robotic framework for picking-up paper (see Section 4.5.2), the resulting hierarchical state machine (HSM) (see Figure 5.20) was mainly realized in a feed-forward manner. The figure does not explicitly visualize the feedback loops of the system as these are either encapsulated within the C_{touch} controller, which is used several times, or natively implemented as active postures that use C_{force} controllers (highlighted as orange states in Figure 5.20).
The system initially registers the paper object in the arm server. The registered object's coordinate frame can then be used as a reference for robot movement commands.
In the following, each top-level HSM state is explained successively. For an optimal connection to the corresponding robot movements, four representative and temporally ordered key-frames are provided for each state. In order to facilitate the comparison of these key-frames, all images are identically cropped.
It is important to note that the course of the interaction sequence is deliberately explained in a very detailed manner in order to emphasize the large set of minor heuristics and *pragmatical fixes* that were necessary to accomplish the task. Although the developed system is currently not capable of solving arbitrary anthropomorphic folding tasks, the principled way in which the system was designed means that, in theory, more complex interactions can be achieved by building upon simpler interaction primitives. The concept of a bottom-up approach that combines *basic action primitives* in order to implement complex actions is presented in Section 6.5.

[10] A video that shows the final robotic folding sequence can be found [Elbrechter et al., 2012b] (second half).

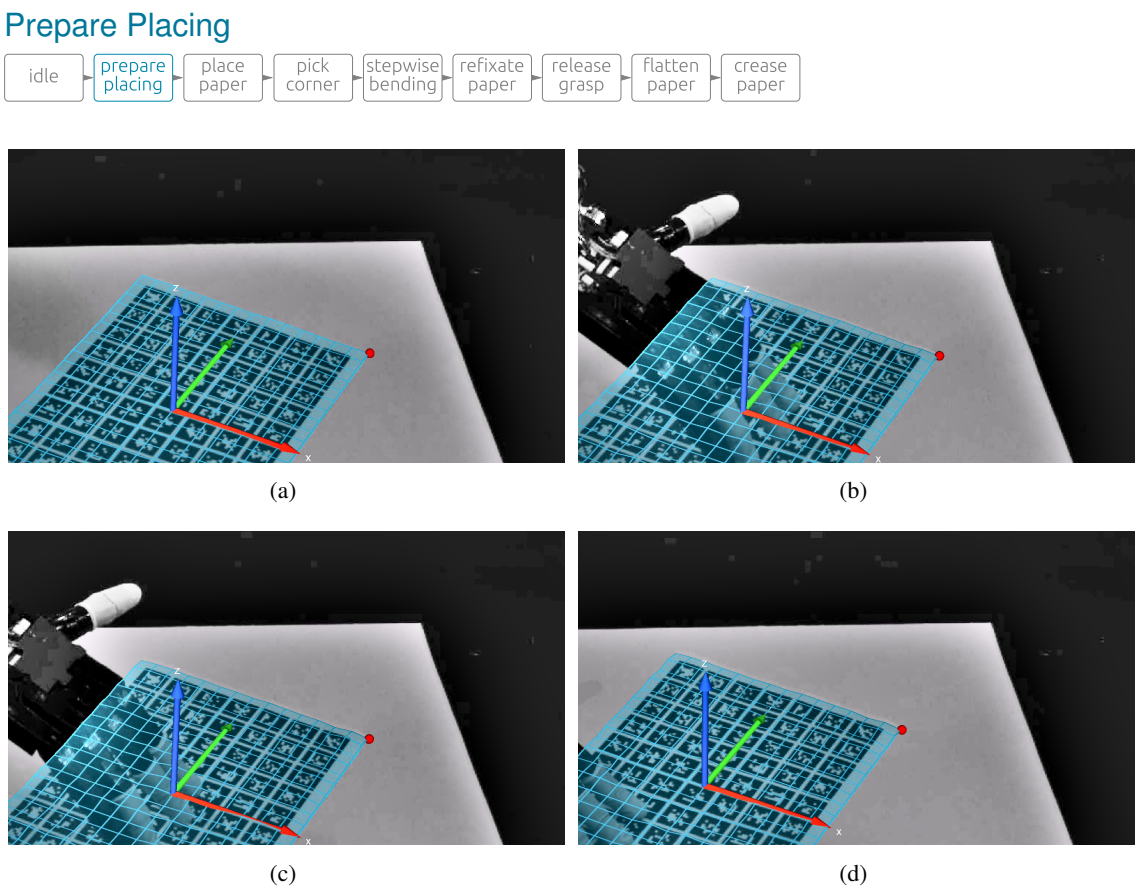


Figure 5.21: **(a)** Initially the paper is placed arbitrarily on the support box and its coordinate frame is estimated. **(b)** The right hand (visible on the left side of the images) hovers relative to the paper coordinate frame. **(c)** By performing a diagonal movement with the hand (upper right to lower left) using the C_{touch} controller, unintended shifting caused by touching the paper is minimized. **(d)** Contact is established; the paper can be shifted.

The system starts in the *idle* state and transitions to the *prepare placing* state when a manual trigger command is received. Here, the system detects the initial position of the paper and establishes contact with the paper. The first sub-state of the *prepare placing* state is *detect*, which is itself implemented as an includable sub-HSM. The *detect* sub-state[11] subscribes to *paper/T* events, which are inserted into the ActiveMemory instance by the 3D estimation and modeling unit (see Figure 5.18a). As soon as a *paper/T* event is received, the transformation of the registered paper reference object is updated and a transition to the next state is performed. During the implementation of the HSM, it first seemed obvious to implement these features by creating an overarching super-state, which concurrently receives *paper/T* events in order to keep the paper object's reference frame always up-to-date. However, tests revealed that performing robot movements with respect to the paper object's coordinate frame while that frame is updated concurrently led to chaotic outcomes. Therefore, updating the paper reference object's transformation with respect to visual feedback was performed only when it is explicitly needed. The subsequent *place-hand* sub-state aligns the robot hand with the edge of the paper using a predefined relative transformation with respect to the registered paper object. Thereafter, an instance of the C_{touch} controller is used to establish contact with paper.

[11] Dependent on the location where its HSM snippet is included to the main HSM file, *detect* can also become a top-level state etc.

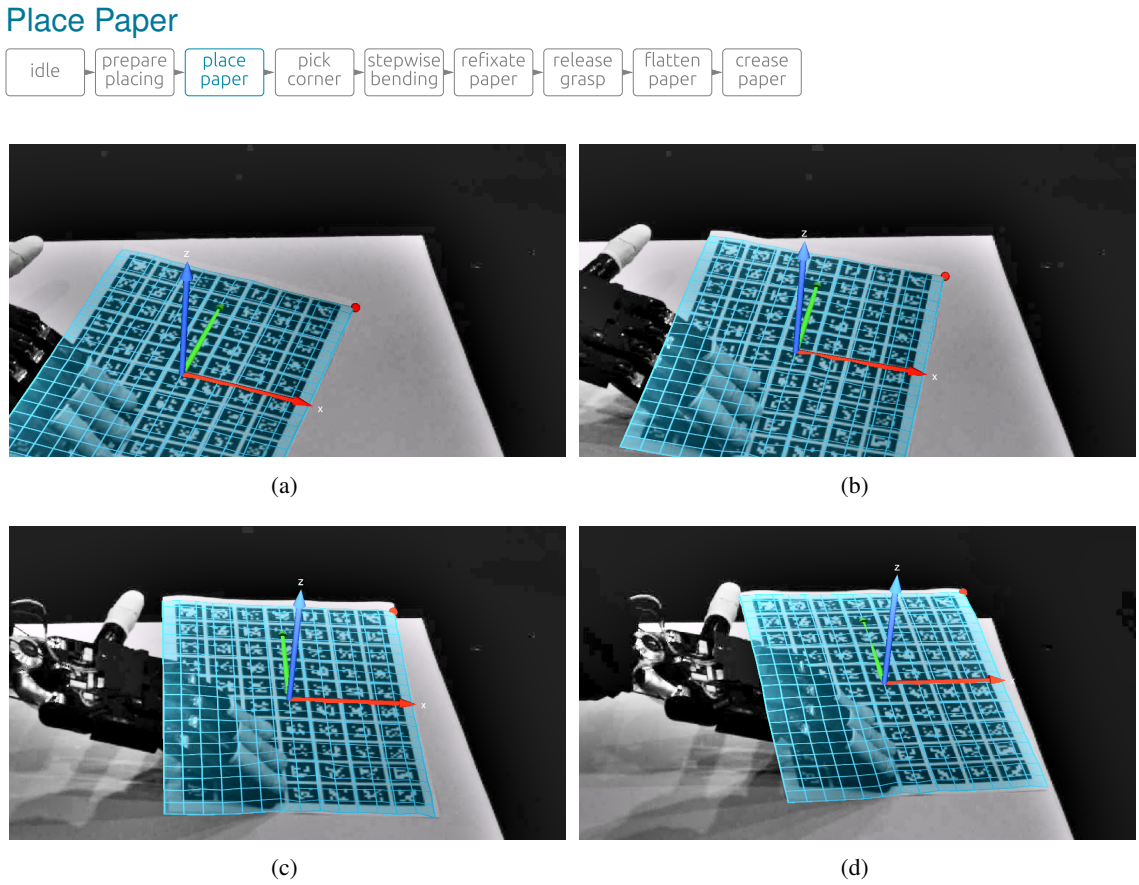


Figure 5.22: The paper is shifted towards the rear right corner of the support box to facilitate pinch-grasping its corner with the other hand (a-d) . While the paper is shifted, the C_{force} controller (implemented as an active posture) is used to maintain a given contact force.

Once contact with the paper is established, it has to be shifted towards an edge of the support box to allow pinch-grasping with the left hand. In order to deal with paper movements that occurred while the hand established touch with the paper, vision-based paper detection is once again triggered to update the paper object's coordinate frame, T_{paper}. The following shifting motion moves the paper to a predefined position and orientation in the world, described by the homogeneous transform T_{target}. This is possible by assuming a well known and fixed position and orientation of the support box. If the support box, or a real table edge, were not previously known they could be detected by visual input, and the resulting target position of the paper would have to be computed dynamically. During the shifting action the paper is now temporarily assumed to be attached to the robot hand. The high friction of the rubber covered fingers avoids slippage. Let T_{robot} be the current robot end effector transform, which is queried in the *plan shift* sub-state, the end target robot effector transform is given by

$$T'_{\text{robot}} = T_{\text{target}} T^{-1}_{\text{paper}} T_{\text{robot}}$$

The resulting shifting motion itself is implemented using an active shifting posture that uses all fingers (except the thumb) for feedback-based control of the contact force. The active posture is not deactivated when the movement is finished. Instead, the controller remains active to fixate the paper while it is pinch-grasped and bent by the other robot hand.

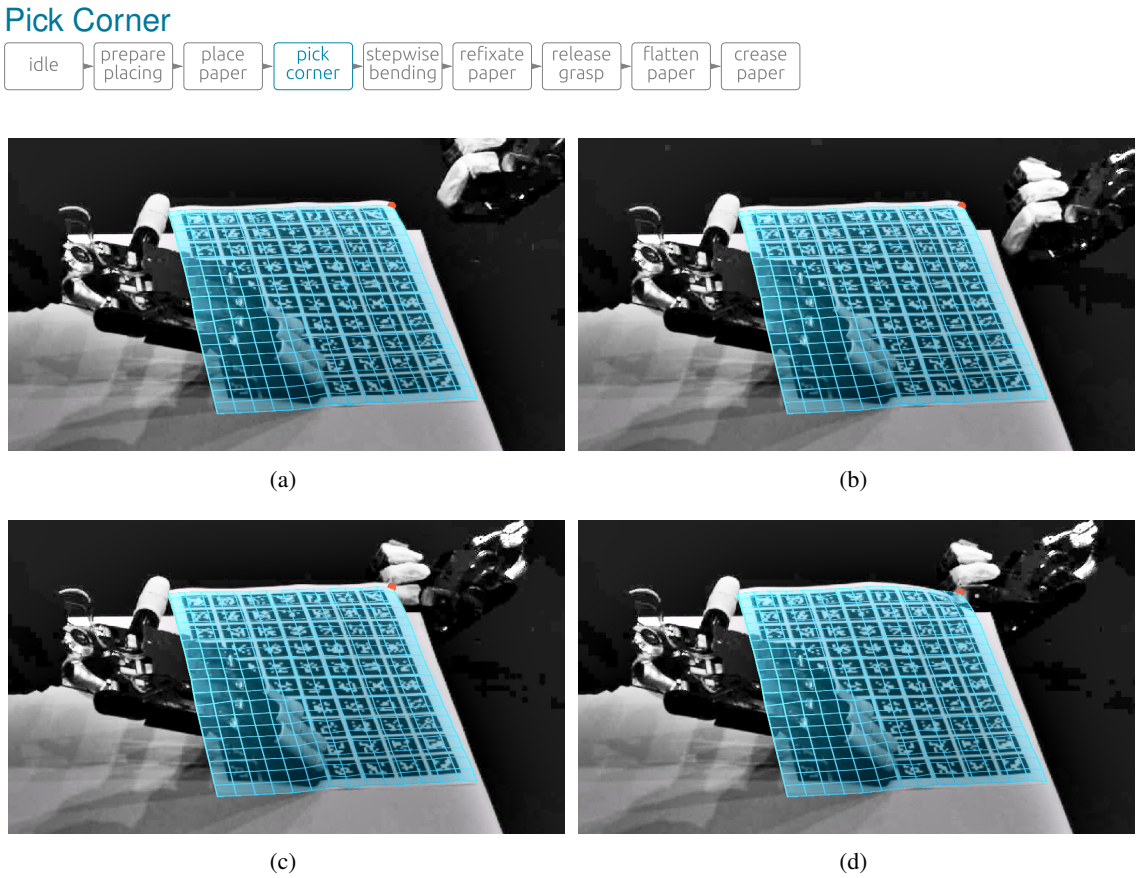


Figure 5.23: While the paper is fixated by the right robot hand, its rear right corner is pinch-grasped by the left robot hand (visible on the right side of the image). **(a)** A fast arm movement is used to approach the paper quickly. In parallel, the pre-grasp posture is actuated. **(b)** A much slower velocity is used for adjusting the z-position to avoid that the robot-server's collision handling predicts a collision between the extrapolated position of the hand knuckles and the table top. **(c)** The hand is moved horizontally to bring the paper corner between thumb and forefinger. **(b)** To grasp the corner of the paper, the final pinch-grasp posture is actuated.

After the paper has been shifted towards an edge of the support box, one of the paper corners can be pinch-grasped by the left hand. Due to the lack of an integrated global planner, the left arm is initially pre-rotated to ensure a desired arm-rotation for the subsequent steps. The compliant hand design, the non-rigid support box and even minor slippage lead to the fact that even after a precise computation of the shifting trajectory, the resulting paper position cannot be predicted accurately. Therefore, the internally used paper coordinate frame is once again updated from the visual feedback. Pinch-grasping is then performed in three steps. In the *approach* sub-state, the left hand is positioned coarsely and the hand's pre-grasp posture is actuated. While the robot movement in this step can be applied very fast, the final approach to touch the paper must be performed much more slowly, as the robot hand is in close proximity to the setup's table top at this point. To avoid damaging the robot hand by erroneously colliding with the fixed parts of the setup, the robot controller uses an internal collision avoidance mechanism that extrapolates robot movements in simulation to avoid hardware collision. The faster the robot moves, the earlier it has to be interrupted to ensure that it can be stopped before a collision occurs. Therefore, the final part of the movement (the last 6cm) is performed very slowly in the dedicated *down* sub-state. Finally, in the *grasp* sub-state, the hand is moved towards the corner of the paper and the final pinch-grasp posture is actuated.

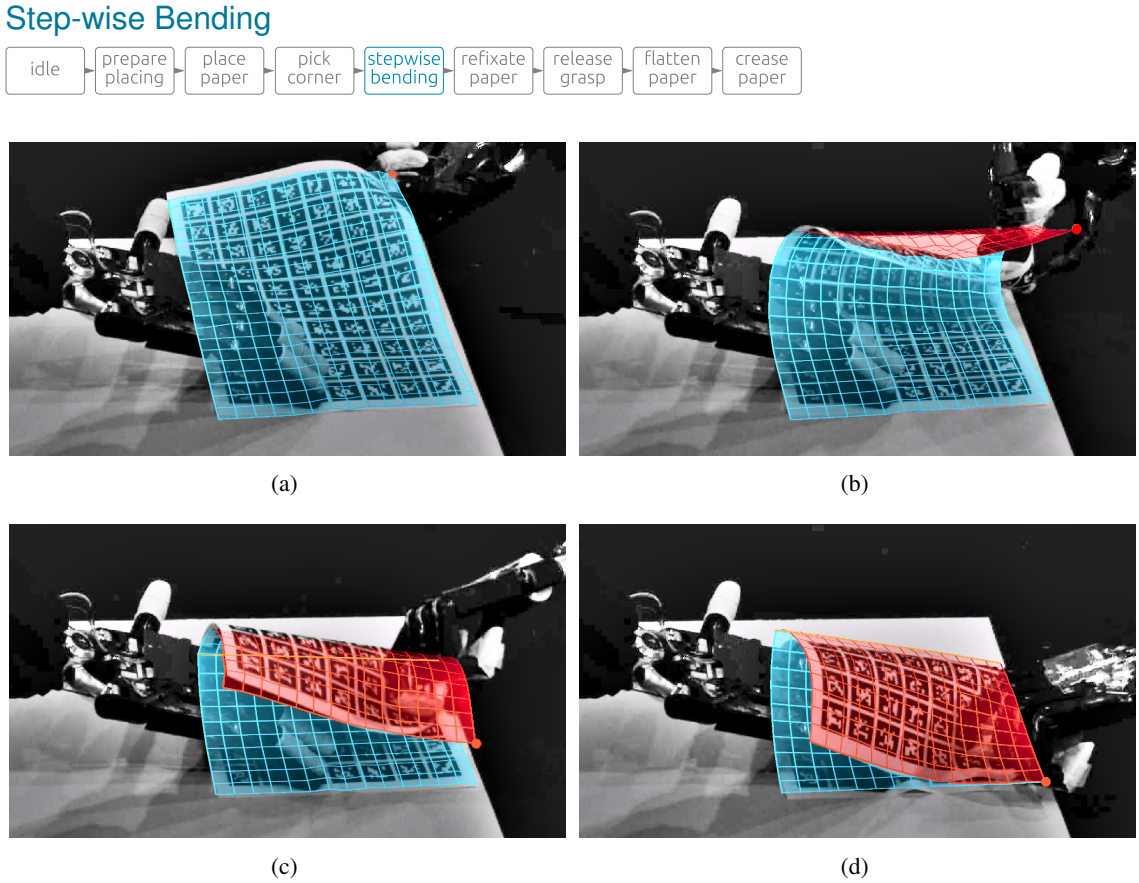
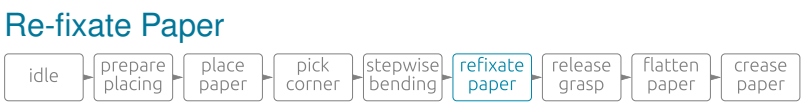
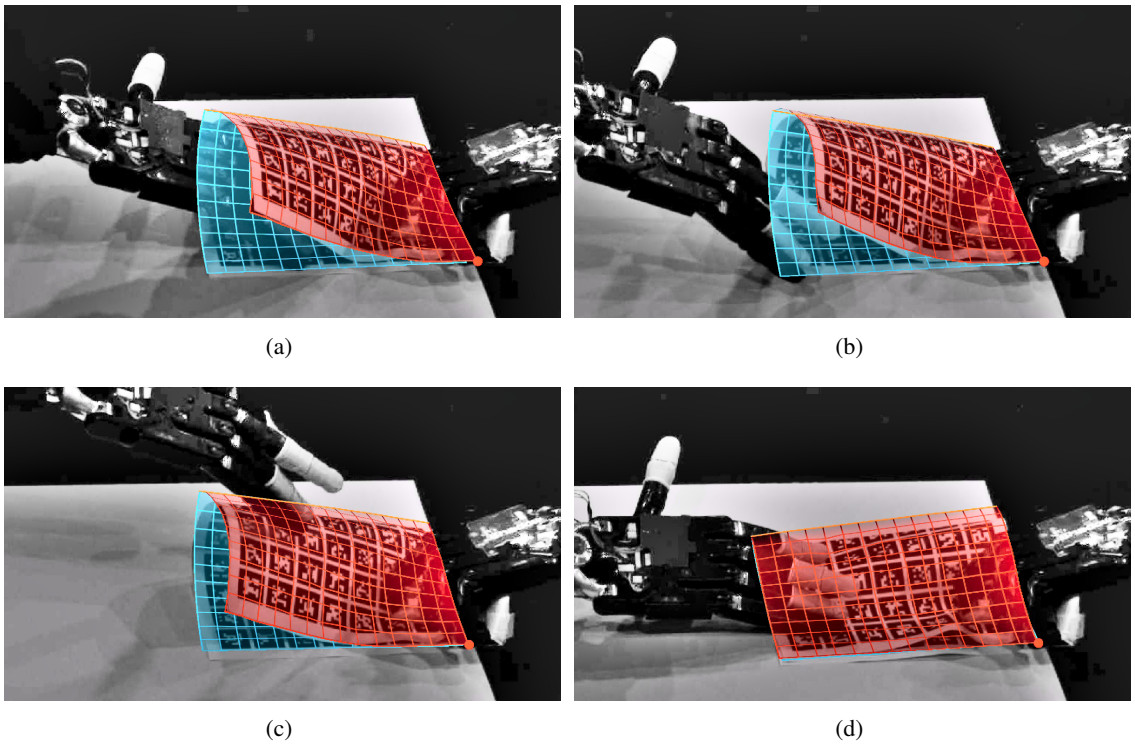


Figure 5.24: The hand-crafted bending motion, defined relative to the detected paper coordinate frame is performed. **(a)** The hand is moved upwards without rotation. **(b)** The paper is bent by simultaneously translating and rotating the hand. This is achieved by interpolating smoothly through a recorded sequence of reference postures. **(c)** The intrinsic maximum model curvature is reached. Therefore the modeling unit is triggered to add a center fold line (orange line) to the physical paper model. **(d)** At the end of the folding sequence, the C_{touch} controller is used to fixate the bottom layer of the paper while the top layer is fixated by the grasp.

After pinch-grasping the corner of the paper, the time has come to execute the bend. An obvious approach for the realization of the bending movement was to implement a circular hand trajectory that is applied in parallel to an appropriate hand rotation. However, the task turned out to be much more complex than initially expected. The practical issue, that the arm server did not support independent translation and rotation movements in a synchronized manner, was solved by performing rotation and translation in small alternating steps. However, also apart from this technical issue the bending trajectory proved to be very complicated to realize programmatically. The desired arm trajectory was neither a circle nor an ellipse, but a rectangular movement with beveled corners. Furthermore, it turned out that the rotation and translation speed need to be decoupled and have to be adapted over time. Together with the fact, that a very special movement was necessary to finally orient the left hand appropriately so that fixating both layers of the paper using the C_{touch} controller was possible while not releasing the pinch-grasp, it was decided to implement the movement by interpolating smoothly within a series of hand-tuned reference postures recorded relatively to the paper coordinate frame[12]. Due to the lack of an automatic fold detection, the detection and modeling unit (see Figure 5.18a) is explicitly triggered to add the fold line to the model at a certain point during the bending motion.

[12] The postures were manually defined by moving the robot with a space mouse.





(a) (b)

(c) (d)

Figure 5.25: **(a)** The right robot hand releases the paper carefully to avoid dislodging it from the left hand's fixation. To do this, the pneumatic muscles are relaxed. **(b)** The right hand is moved out from the space between the two layers of the paper. **(c)** The right hand is appropriately re-positioned on top of the paper. **(d)** Finally, both paper layers are fixated using the C_{touch} controller.

When the system transitions to the *refixate paper* state, the paper is temporarily fixated by both robot hands. However, at this point neither of the hands fixates the paper appropriately for flattening with the other hand. The right hand fixates only the bottom layer so that releasing the left hand would result in the paper unbending immediately. In contrast, the left hand fixates both paper layers, but only the top layer, which is still pinch-grasped, is firmly fixated. The fixation of the bottom layer is not only weak due to the decreased friction of the hand's fabric finger covers, but also because of the fact that the left hand touches that layer of the paper with the outer finger surfaces only. A human would easily adapt the fixation posture to achieve a more stable fixation center, but such a movement was not possible with the robot hand. Therefore, the paper fixation by the right hand is updated to fixate both layers of the paper properly. To this end, a trivial sequence of hand-crafted and manually tuned movement primitives, defined with respect to the paper frame, was used. The initial *release* sub-state employs the knowledge about the difficulty of *carefully* releasing the paper gained in the picking up experiment (see Section 4.5.2). In the following *arm up* state, the robot arm is first slightly moved upwards and then horizontally away from the paper to minimize contact with the upper layer of the paper. Before the hand is positioned appropriately over the paper to prepare the re-fixation in the *reposition* sub-state, the paper coordinate frame is updated from visual input. The final fixation is again implemented using the C_{touch} controller.

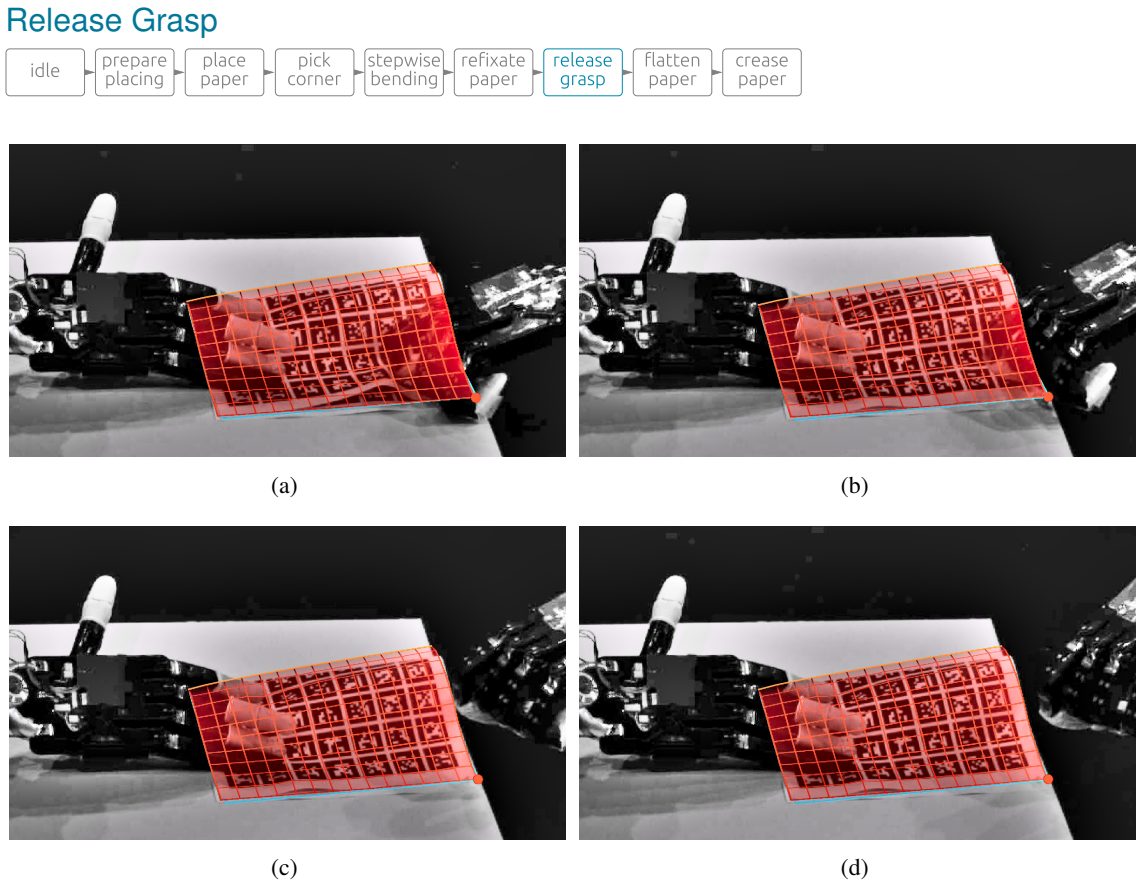
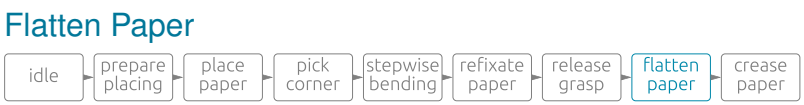
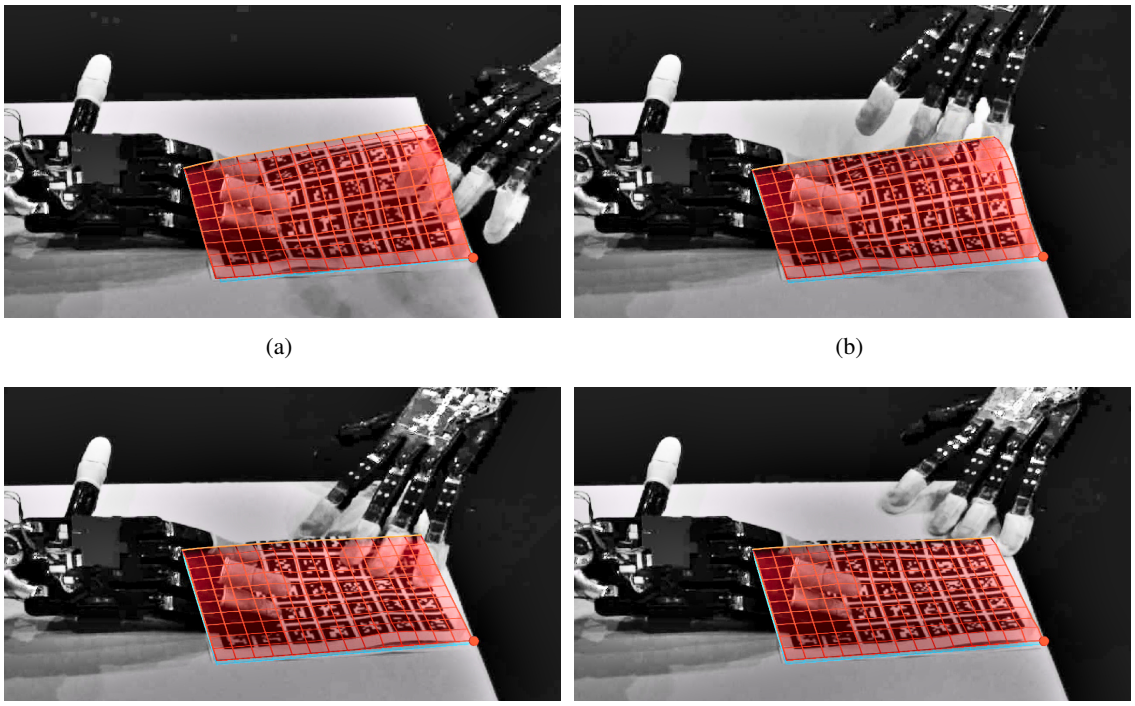


Figure 5.26: The complex fixation of the paper by the left robot hand is undone **(a,b)** Carefully releasing the paper starts with relaxing the hand's pneumatic muscles. This not only releases the top layer from the grasp, but also the pressure-based fixation of the bottom layer. **(c,d)** The left hand is moved away from the paper.

Once both layers of the paper are firmly fixated by the right robot hand, the left hand is prepared to transform the bent center of the paper into a fold. To this end, first the right hand is released and the arm is moved upwards. Releasing the hand is particularly complex here since the upper layer of the paper is still firmly grasped and held down. However, releasing the pneumatic hand muscles before moving the arm helps to minimize unintended pulling or shifting of the paper.





(a) (b)

(c) (d)

Figure 5.27: The bent paper is flattened by the left hand in order to facilitate creasing. **(a)** The left hand was released from the paper and the paper coordinate frame is queried from the modeling unit. **(b)** The hand is positioned over the paper. A downwards movement is initiated to establish contact using the C_{touch} controller. **(c)** Once contact is established, the active posture controller C_{force} is used to maintain contact pressure while swiping over the bent paper center. **(d)** Swiping over the bent center results in a weak crease.

Making a fold into the sheet of paper is realized in a two step manner. Before a precise creasing operation can be applied to harden the paper's fold line, the fold is prepared by swiping with a low force over the bent paper. The current paper coordinate frame is once again queried. After positioning the hand appropriately with respect to the detected paper coordinate frame, a combination of the feedback-based controllers is used to realize the swiping motion. First, C_{touch} is used to establish contact using a low posture difference threshold, resulting in a weak contact force. Subsequently a hand-crafted swiping motion is performed while an active posture is activated. The pattern of this combination of the feedback-based controllers matches the method used for initially shifting the paper. However, while shifting was performed with the high-friction rubber covered fingers of the right robot hand, the swiping motion is performed with the low-friction fabric covers of the left hand and the paper is simultaneously fixated by the right hand. At the end, the left hand and the arm are released and moved upwards to prepare the robot for the final creasing motion.

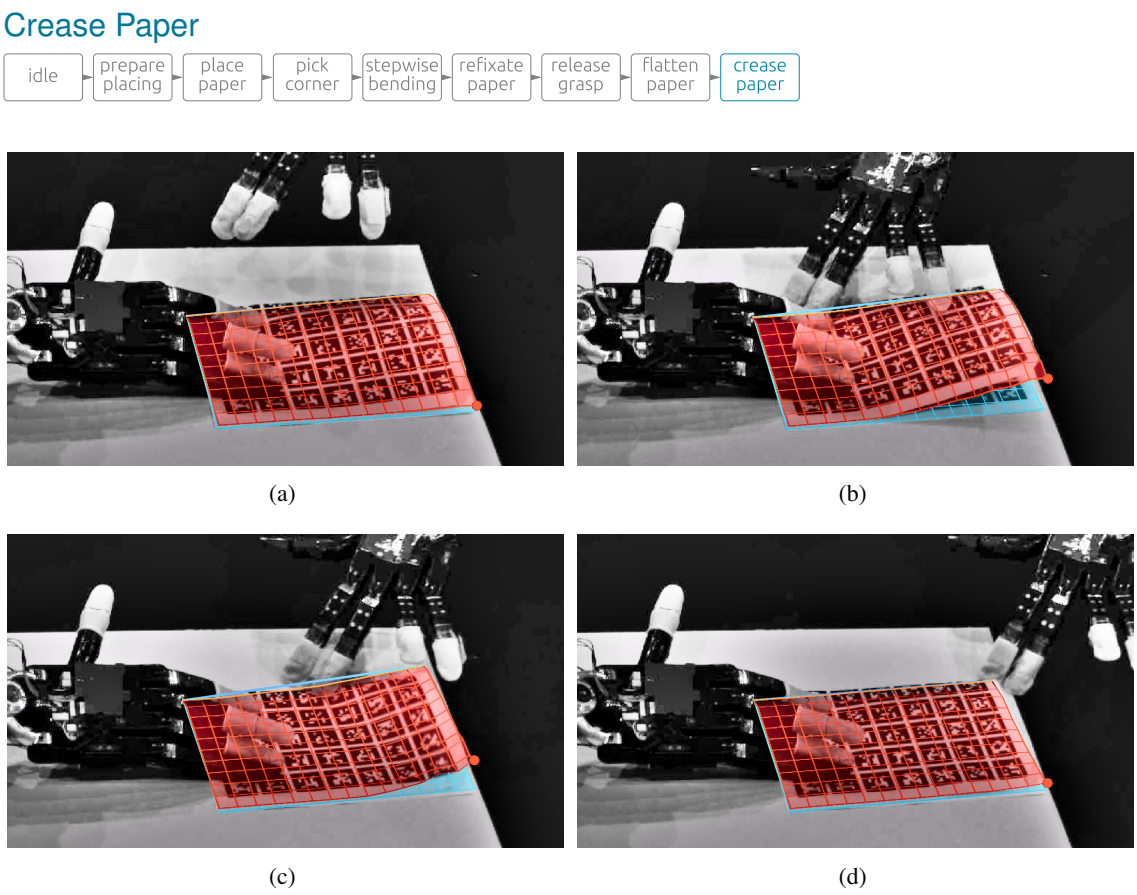


Figure 5.28: The weakly flattened crease is hardened by a precise two-finger creasing motion of the left hand. **(a)** Initially, the coordinates of the physical model's fold line is queried (blue line, in (a) hidden by the orange line) from the detection and modeling unit and the hand is placed. **(b)**The C_{touch} controller is configured to measure the joint displacements of the two used fingers only. **(c)** As soon as contact is established, the C_{force} controller is used to maintain the contact force during the creasing motion. **(d)** Once the creasing motion is accomplished, the paper is folded in half.

Creasing the paper is achieved in a very similar way to the preceding flattening step, but with a few noticeable differences. In the *crease paper:detect* sub-state, not the paper coordinate frame *Paper/T*, but the 3D position and the orientation of the approximated crease-line *Paper/C* is queried from the detection and modeling system. Once the coordinates of the fold line are available, the actual creasing motion is performed with the robot's fore and middle fingers. Again, both feedback-based controllers are used here. The end result of the folded paper after both robot hands have been removed is shown in Figure 5.29. The folding was successful, but the final creasing did not fully harden the fold on the left hand side.

5.6 Discussion

In this chapter, the formerly presented robot system for picking up paper (see Section 4.5) was extended and improved in order to realize the more complex task of bi-manually folding a sheet of paper in half. To realize this, extensions along several axes were necessary.
By improving the marker-based visual detection of key-points, the system is much better able to deal with even severe occlusions, which naturally occur very frequently during the manipulating

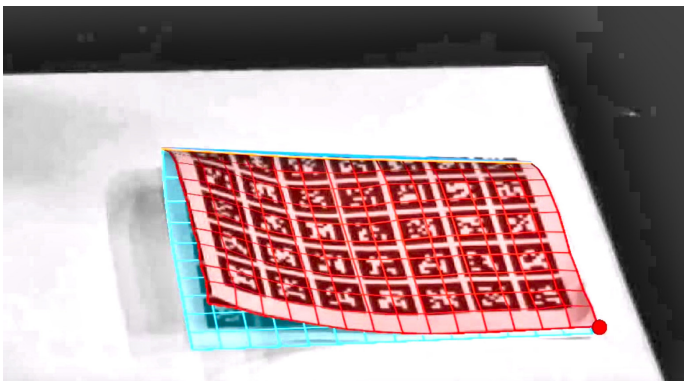


Figure 5.29: End result of the robotic folding sequence. The paper is well folded, but the hardening of the fold was not fully successful on the left hand side.

of paper. The implemented detection engine for BCH-markers was shown to significantly outperform the previous markers in terms of the maximum detection distance and the error rate for marker identification. The more flexible physical model, which is able to represent crease-lines and to memorize deformation of the paper significantly increases the complexity of paper configurations that can be modeled and tracked by the system. The capabilities and the limitations of the system were evaluated on the basis of several common paper folding experiments carried out by a human. By introducing a new vision-guided model control law that is able to move arbitrary model coordinates, not only was the present marker-based key-point detection mechanism integrated, but also a major prerequisite for the potential use of other more general and marker-less key-point estimation mechanisms was provided for. Furthermore, the performance of the whole vision-based detection and modeling system was optimized so that it can deal with at least 6 1-mega-pixel cameras running at 15Hz on a single PC in real-time.
For the realization of the robotic experiment, two versatile feedback-based controllers were developed and integrated. Their value was demonstrated by deploying them several times during the robotic paper-folding sequence. The C_{touch} controller, which is used for joint-feedback based contact establishment, was used in several very different hand-object configurations. It proved to be useful not only for common four or two finger based touching, but also when establishing contact while a grasping posture is actuated. The tactile feedback based controller for maintaining a certain contact force, C_{force}, was also successfully used for different tasks, such as shifting, fixating or creasing the paper.
The final robot control system was successfully used to fold the paper in about 80 seconds and through an extensive manual tuning process a final overall success rate of 80% (16 our of 20 trails) was achieved.

5.6.1 Visual Detection

Even though, the improvements applied to the marker detection system were very satisfying, the necessity to densely cover the paper with fiducial markers on both sides is one of the system's major drawbacks. While a possible alternative would be to use invisible to the human eye IR-reflective ink markers with appropriately filtered cameras, this overhead is not something we seriously considered. Furthermore, fiducial marker detection cannot simply be exchanged by the use of common generic image features, such as SURF [Bay et al., 2008] or ORB [Rublee et al., 2011] as such generic image features are very prone to errors [Rublee et al., 2011]. While these are commonly filtered out using RANSAC [Fischler and Bolles, 1981] based methods, the assumption is that objects are rigid, which is not the case for highly deformable paper. Alternatively, 3D

point-cloud-based detection of deformed paper seems possible, but common ICP [Zhang, 1992]-based rigid body tracking algorithms need to be adapted and several performance issues have to be solved. In addition, the segmentation of the paper object from the whole point-cloud becomes very difficult, in particular in presence of severe occlusions by the manipulating robot or human hands. In order to provide a better idea of possible marker-less detection and tracking of the deformation of paper, a prototype system was developed and its potential was qualitatively evaluated in the Sections 6.2 and 6.3.

5.6.2 Modeling

The physics-based model of paper was shown to provide the flexibility needed for the given task. One of its major drawback is, however, the inaccurate approximation of diagonal folds. Due to the fact that the grid and constraint structure of the paper is static and only the stiffness coefficients of existing bending constraints are altered to model foldable parts of the paper, a relatively high number of initial model grid cells is needed. In order to make the resulting denser grid of soft body nodes behave like paper, a very large set of bending constraints is needed, which makes the modeling system easily hit the computational limits on current hardware. In Section 6.1, a generalized paper model is presented in which the paper is no longer defined by a regular grid of nodes, but by an irregularly triangulated and bounded 2D surface in 3D space. When a fold line is added to the paper model, each existing paper triangle that is intersected by that line (in the 2D paper model space) is broken into three further triangles. By this mechanism an intended fold can be modeled precisely by a line. In turn, this allows the initial triangulation of the paper to be much more coarse.
Another drawback of the detection and modeling engine is the fact that crease lines have to be added manually to the model. A more elaborate system could start with a coarse paper grid that is then automatically refined locally along automatically detected fold lines. For automatic fold line detection several promising approaches, such as incorporating local curvature approximation or even particle based systems, in which each particle represents a whole fold configuration of the paper, could be considered. In order to provide a better understanding of the potential and the limitations of such a system, a prototype was developed and coarsely evaluated (see Section 6.4).

5.6.3 Robot Control

Even though, the extensions to the presented detection and modeling engine were shown to be well suited for monitoring even very complex paper manipulations, the task selected for the robot was explicitly chosen to be simple. The main reason for this was the uncertainty of how to integrate vision, modeling and proprioceptive robot feedback in a more complex interaction scenario. Instead, a simpler task allowed the essential aspects to be explicitly put into focus. In particular we wanted to explore contact situations with the paper, which led us to the development of the generic closed loop feedback controllers C_{touch} and C_{force}. However, while these controllers allowed the paper-folding task to be carried out in a sufficient manner, the experiment also revealed the limitations of the hard and software setup.
The Shadow dexterous robot hands were not only very difficult to maintain in good working order, but an increase in their speed and pose-accuracy (in both actuation and proprioceptive measurement) would have contributed significantly to the quality of the final demonstrator. An adaption of the system to the newly available Shadow motor hands [Shadow Robot Company, b], would solve most of these issues. However, while the passive compliance of the used Shadow dexterous hands reduced the likelihood of the hands becoming damaged, using motor hands would necessitate a detailed active modeling of their compliance properties.

The system for bi-manual paper folding was able to react to many dynamic factors, mostly resulting from the imprecise shape prediction of the deformed paper. Despite its flexibility, most parts of the manipulation sequence had to be hard-coded. Possible future extensions range from more dynamic handling of proprioceptive, visual and haptic feedback, over autonomous and hierarchical planning of paper manipulation patterns to learning from demonstration or even autonomous exploration.
The deformable but plastic behavior of manipulated paper generates high demands on tactile sensors needed for the robot hands. While humans can detect forces with their finger tips ranging over many order of magnitudes, tactile sensors, small enough to fit into a robot finger tip and yet able to robustly recognize contact with a part of a piece of paper are still not available [Kõiva, 2014]. Given sensitive enough finger sensors, the demand for the visual feedback could be significantly relaxed. In turn, this would also allow more complex in-hand manipulation that does not rely necessarily on the counter-force provided by the table-top to be performed. In an optimal system, the robot's *hand-eye-calibration* would be *smart enough* to seamlessly join both visual and tactile information streams, allowing the system to dynamically rely on the type of information that is given with a higher accuracy, relevance or certainty.
Finally, we note that the system does not use autonomous planning at all. The main reason for this is that it is not supported natively by the used robot control architecture. As discussed in Section 5.1, planning of interaction with compliant objects, in particular paper, is a very complex task that is thus far not even close to being fully solved in all of its aspects. By integrating the Bullet-Physics engine into a robot planning framework, the paper model could be used for planning without a large overhead. However, it is important to mention that the integration of physics into robot planning has until now not been completely solved as it requires the accurate tuning of physics parameters, which is even difficult for the case of rigid objects [Weitnauer et al., 2010].
Further concepts regarding the robot control for anthropomorphic manipulation of paper and other paper-like objects are comprehensively discussed in Section 6.5.

6 Advanced Aspects

This chapter provides some insights into possible future extensions and indeed general directions the paper detection, modeling and manipulation framework could proceed in. While thus far, each application chapter contained a whole iteration cycle along each of the axes *detection*, *modeling* and *robot control*, the aspects presented here are treated separately and are not integrated into *final* demonstrator systems.

As discussed in Section 5.6.2, the regular grid structure of the original paper model shows a poor modeling accuracy when modeling diagonal folds. Therefore, a more sophisticated physical paper model was developed and implemented that no longer relies on a predefined and regular grid structure. The new model's surface is represented by an unordered set of triangles, whose corner vertices define the link between a position in the world and an arbitrary position on the bounded 2D surface in space. Instead of modeling folds by adapting the stiffness coefficients of intersecting bending constraints, the new model automatically splits intersected triangles in order to represent the fold line precisely in a geometrical manner. The model is described and qualitatively evaluated in Section 6.1 and it is used as a basis for other extensions presented in this chapter.

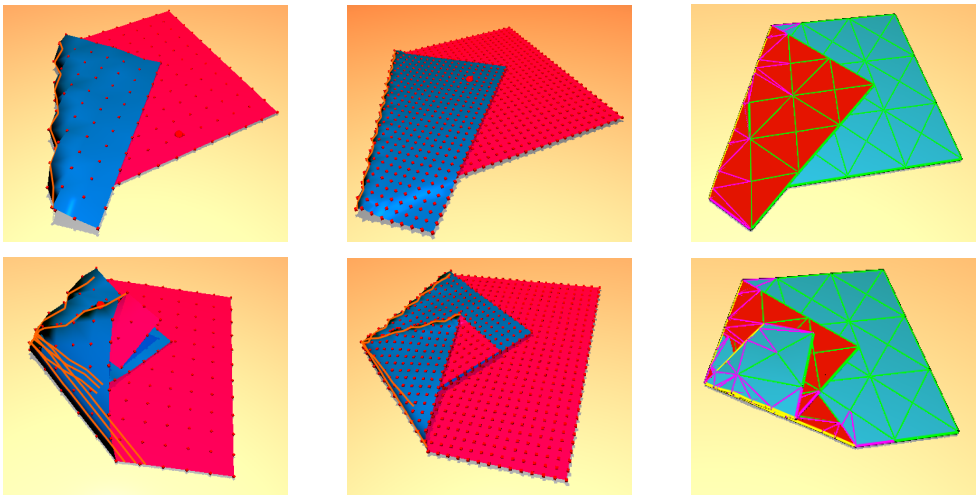
The strongest negative remark received concerning the detection system was the necessity to densely cover the sheet of paper with fiducial markers. Due to the availability of the Microsoft Kinect camera, the obvious question is whether the marker-based detection could be replaced employing point cloud processing methods – if necessary, supported by standard 2D-image feature detection. In order to provide a better understanding of the possibilities, difficulties and realistic limitations of such an approach, a marker-less point-cloud-based test system for paper-tracking was designed, implemented and evaluated in a qualitative fashion. As an initial test, a trivial extension of standard iterative rigid-body tracking methods is presented and a set of possible heuristical optimizations is evaluated (see Section 6.2). This leads to the insight that the major drawback of a missing static and reliable association between paper-surface points and points in the point-cloud cannot be fixed without incorporating more complex features. Therefore, an additional extension that combines classical Iterative Closest Point (ICP)-based tracking with 2D SURF-features that are mapped into the input point-cloud is introduced (see Section 6.3) and its tracking performance is compared to the original marker-based tracking.

Another drawback of the existing system is the fact that folds have to be added manually, i.e. both, the temporal onset of adding a fold as well as the new fold's geometry have to be provided. A system that allows folding actions to be detected automatically would significantly contribute to the reduction of the needed user input. However, we note that when embedded into a robotics system, the temporal onset of the adding of a fold can be easily obtained as the robot *knows* when it starts to fold the paper. In contrast to this, the actual fold geometry is extremely likely to differ at least slightly from the intended geometry. Therefore, a service that automatically detects the actual fold geometry would allow the robotic system to decide whether a folding action was carried out successfully or whether it has to be redone. A difficult sub-problem here is to distinguish between *heavy bending* of the paper and actually folding it. As this can become arbitrarily complex to differentiate if the system's input is only the current paper configuration, two extended approaches were considered. The first idea was to not only track the deformation of the paper, but also the movements of the manipulating hands. From this, the system would be able to much more easily detect when the paper is actually folded. However, since this entails the necessity of being able to track the manipulating human hands, an alternative method was investigated, developed and

evaluated. In certain situations, when it seems probable to the system that a fold line was added, a particle system is initialized, where each particle represents a fold (or non-fold) hypothesis linked to a separate instance of the paper model. In the following processing steps, these potential models are used in parallel for tracking the ongoing paper manipulation. After some time, the model that is calculated to be closest to the real world situation is assumed to be correct and the other particles are discarded. Of course, it has to be admitted that this leads to a significantly increased computational complexity that scales at least linearly with the number of particles spawned. Furthermore, the testing system is thus far not implemented in a hierarchical manner which is a necessary step if the system is to track several creasing operations in parallel. Some extensions towards automatic fold detection and a discussion about the limitations of straight-forward approaches are provided in Section 6.4.
The chapter ends with providing a theoretical concept of a generic robotic system for anthropomorphic manipulation of paper and other 2D deformable objects (see Section 6.5). The system is conceptualized in a bottom-up approach, that is, it is based on an extendable set of *basic action primitives* (BAPs) that are sequenced, combined in parallel or hierarchically cascaded to realize more complex actions. The initial set of BAPs is created by monitoring an interaction sequence in which a human folds a paper aeroplane. Subsequently, the set is employed to theoretically realize other increasingly complex manipulations, such as *flicking through the pages of a book*, *folding a piece of cloth* or *putting a slice of cheese on a slice of bread*. This allows us to continuously refine our set of BAPs by extending and generalizing the existing primitives or by adding new ones. Furthermore, by considering examples, which are progressively less related to paper, the generalizability of the system is tested and tuned.

6.1 A Generalized Paper Model

The poor modeling accuracy of diagonal folds was shown to be a major drawback of the suggested paper model (see Section 5.6.2). It not only leads to aliasing effects along non-axis-aligned fold-lines (see Figure 6.1a), but also accounts for modeling difficulties in cases of iterative folding (see Section 5.4). If a regular grid of nodes is kept, the only way to decrease the aliasing effect along diagonal fold lines is to use a more fine-grained grid structure (see Figure 6.1b). However, this not only leads to computational disadvantages, as the model's constraint structure scales quadratically with the number of nodes, but it also affects the overall modeling quality. The main reason for this is the fact that a more fine-grained grid surface implicitly behaves less stiffly and therefore more cloth-like in the Bullet physics engine. This effect can be compensated by increasing the number of bending constraints and their stiffnesses, but only to a certain degree at which point a further increase of link stiffness coefficients negatively effects the numerical stability of the physics simulation.
Therefore, in this section, a generalized paper model is presented (see Figure 6.1c). In contrast to the model used for the folding experiment (see Section 5.3), the new model is no longer limited to a regular grid of nodes. Instead, the generalized paper model is distinguished by a set of nodes $n_i = (\mathbf{x}_i^m, \mathbf{x}_i^w)$ $(i = 1, .., N)$, each linking an arbitrary position $\mathbf{x}_i^m \in P$ on the paper surface to a current world position $\mathbf{x}_i^w \in \mathbb{R}^3$. Note that $P = [0, W] \times [0, H]$ is the bounded 2D paper manifold (see Section 4.3.1) and the paper's undeformed size is assumed to be $W \times H$. The nodes are connected by a set of triangles $t_i = (t_i^1, t_i^2, t_i^3)$ where each $t_i^j \in \{1, .., N\}$ indexes a certain node. The triangle set is defined in such a way, that **i)** P is fully covered by triangles and **ii)** triangles never intersect. The paper surface function $p : P \rightarrow \mathbb{R}^3$ is defined by piece-wise bi-linearly interpolating between the world positions referenced by the triangles' corner indices.



(a) Old model (coarse grid) (b) Old model (fine grid) (c) New model (dynamic)

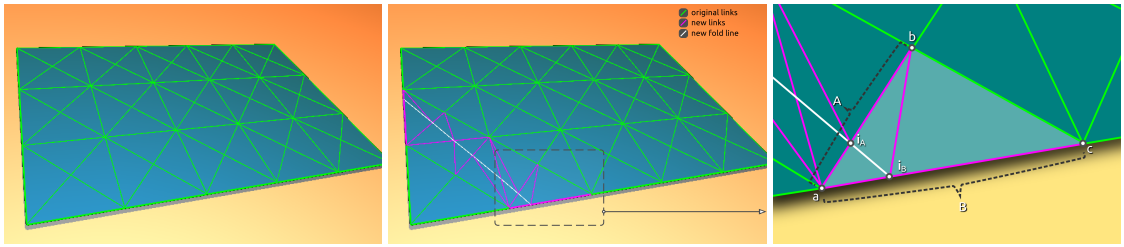
Figure 6.1: Comparison of the fold modeling accuracy between the original model (a,b) and the new model (c). After performing an initial diagonal fold (top row), a second intersecting fold is added (bottom row). **(a)** The coarse grid structure (12×15 nodes) leads to a strong aliasing effect along the fold. The resulting model tension limits the maximum fold curvature so that the fold appears not as sharp as intended. In turn, the surface stiffness remains too *refractory* to allow a second intersecting fold to be added unless all constraints within an area around the fold are also weakened by adding several additional parallel folds (bottom a). **(b)** A fine grid structure (30×40 nodes) allows folds to be modeled more precisely but the increased computational complexity is no longer real-time tractable. **(c)** The new generalized model features a dynamic grid structure. Thus, folds no longer have to be approximated, but they can be represented by real geometry. By these means, even with a very small initial node count (here 39) both, diagonal and intersecting folds are represented very precisely.

6.1.1 Constraints

In a similar fashion to the formerly presented paper models, the physical soft-body stiffness of the generalized model is defined and controlled by constraints c_{ij} that define a desired distance $r_{ij} \in \mathbb{R}^+$ of the two referenced nodes n_i and n_j. An additionally attached stiffness coefficient $s_{ij} \in [0, 1]$ allows us to control the *weight* of the constraint. Again, we have to distinguish between constraints that connect adjacent nodes (distance preservation constraints) and those spanning over one or more other nodes (bending constraints). However, in contrast to the model used for the picking-up experiment, the adjacency of nodes is no longer defined by a minimal city-block distance in the discrete paper model grid, but by the edges of the triangles associated with the paper-model. To this end, for each triangle t_i, the 3 edges $\{(t_i^k, t_i^l)|k, l \in \{1, 2, 3\}, k \neq l\}$ are treated as distance preservation constraints. Since adjacent triangles can share an edge, doubled or reversed links are discarded internally. The missing regular grid of nodes leads to the same issues for systematically creating bending constraints, which were before also added using the city-block metric. Instead, bending constraints are now added on the basis of the pairwise 2D Euclidean distance of two nodes in the paper space.

6.1.2 Folds

Adding a fold line to the paper model was from a technical point of view, one of the most complex and difficult features to implement (see Figure 6.2). In a similar fashion to the formerly presented testing applications, fold lines are added manually to the model using a mouse-based drag-and-



(a) Initial paper model (b) Model after fold line added (c) Magnification of one triangle

Figure 6.2: Adding a new fold line to the paper model by splitting existing intersecting triangles **(a)** Shows the original paper model that is initialized with a regular triangle structure. **(b)** Adapted paper model. Each triangle that intersects with the fold line is split into 3 smaller triangles. **(c)** Magnification of a single split triangle. The two intersected edges, A and B, of the triangle are cut at the points $\mathbf{i}_A$ and $\mathbf{i}_B$.

drop gestures. As a simplification, the splitting of triangles can be performed in the 2D projection space of a virtual camera, which allows us to perform the splitting in 2D rather than in 3D space, however, the implemented method can easily be extended.
The mouse gesture yields a 2D line L. The split of the paper along that line can basically be performed on each triangle separately. To this end, the paper nodes are first perspectively projected using the visualization's current view camera parameters, yielding the paper as a set of 2D-triangles that can be intersected efficiently with L. In general, a 2D triangle is either split by L into a triangle and an irregular quadrangle or it is not intersected at all. Without loss of generality, the split intersects two lines $A = \overline{\mathbf{ab}}$ and $B = \overline{\mathbf{ac}}$ of a triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ (see Figure 6.2c). The split ratios of A and B, r_A and r_B, provide the intersection points $\mathbf{i}_A$ on A and $\mathbf{i}_B$ on B. These are defined by

$$\mathbf{i}_A = \mathbf{a} + r_A(\mathbf{b} - \mathbf{a}) \quad \text{and} \quad \mathbf{i}_B = \mathbf{a} + r_B(\mathbf{c} - \mathbf{a}).$$

Hence, the problem of finding the intersection between the line L and the triangle can be reduced to finding the intersection of two line segments (pairwise, L with $\overline{\mathbf{ab}}$, $\overline{\mathbf{ac}}$ and $\overline{\mathbf{bc}}$), which can be implemented efficiently using standard 2D geometry methods.
Due to the law of the preservation of distance ratios under projective transforms, the splitting factors r_A and r_B can directly be used for the splitting of the model triangles that are defined in the model space P. While the adding of a fold lines requires mostly trivial adaptions to the first order links (i.e. distance preservation constraints) of the model which basically reflect the model's triangulation state directly, the adaption of the bending constraints is much more complex. In particular, if the system is to adapt the stiffness of a certain fold in a subsequent step of the manipulation sequence, added fold lines have to be stored internally. By these means a fold's current geometry can be memorized. This was implemented using a discrete 2D *fold map*, which memorizes the fold state of each very small (i.e. 1mm^2) surface patch of the paper model. The fold map function $C : P' \rightarrow [0, 1]$ approximates the theoretical bending stiffness of the paper at a given 2D model position, where P' is the discretized version of the model space P.
Whenever a fold is added, or its stiffness is altered, it is memorized in a discrete fashion in the fold map C. After each adaption, all bending constraints are re-created and their stiffness coefficients is set to the minimal fold map entry that intersects with the link (see Figure 6.3g-j). As an alternative to this, folds could also be memorized in a geometrical representation, which, however, leads to several non-trivial numerical issues when deriving a constraint's stiffness from the set of *intersected* fold lines. The presented technique that is based on the 2D projection has several indirect advantages and disadvantages. It's major drawback is the fact that it only guarantees completely straight fold lines if the paper is fully flat. In other cases (see Figure 6.3a-c), a bend model can lead to also curved fold lines that do not allow the paper to be actually folded.

However, in contrast to this rather negative side effect of the method, it also provides a set of implicit advantages that allow for a very natural and intuitive mouse based adaption of the model. In order to force straight fold lines, the system could compute the intersection of the mouse line L with the paper edges only and then perform the splitting along the straight connection line in the undeformed 2D paper model space. However, tests showed that such a heuristic feels far less intuitive and even sometimes leads to completely unpredictable behavior, in particular, when the model edge is, due to its deformation, intersected more than twice. In addition, the presented interaction method mimics the behavior of real paper very well when many layers are folded on top of each other (see Figure 6.3d-f).

6.1.3 Moving the Model

The movement of the model is important for both, mouse based interaction with the model as well as for moving the model to reflect incoming vision based input. Due to the lack of a regular grid structure, the original control law (see Section 5.3.2) had to be extended in order to be able to deal with the new model structure that is based on irregular triangles. In contrast to moving the nodes defining the model quadrangle that contains the model position $\mathbf{p}^m$ that is to be controlled, the new control law moves all nodes in the vicinity (in the paper space) of $\mathbf{p}^m$. Again, the movement is applied using velocity based control in the physics engine. However, rather than using $\alpha_i = \max(1 - \|\mathbf{p}^m - \mathbf{x}_i^m\|, 0)$ (see Equation 5.2) as a node-specific weighting factor for the movement amplitude of node n_i, a Gaussian

$$\alpha_i = g_\sigma(\|\mathbf{p}^m - \mathbf{x}_i^m\|) \tag{6.1}$$

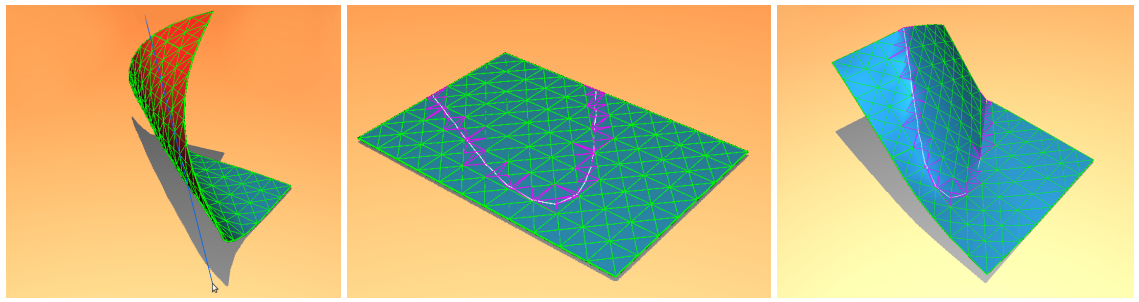
is used to model the locality of the movement law. An internal cut-off threshold α_{min} that avoids moving nodes using negligible velocities is only used for performance optimization. Thus, based on the 3D displacement $\mathbf{d} = \mathbf{t}^w - \mathbf{p}^w$ between the *to-be-moved* paper surface position $\mathbf{p}^w$ and the target position $\mathbf{t}^w$, the P-controller for the velocity-based control of node n_i is defined analogously to Equation 5.1:

$$\mathbf{v_i} = \begin{cases} \alpha_i \lambda \mathbf{d} & \alpha_i > \alpha_{\text{min}} \\ \mathbf{0} & \text{else} \end{cases} \tag{6.2}$$

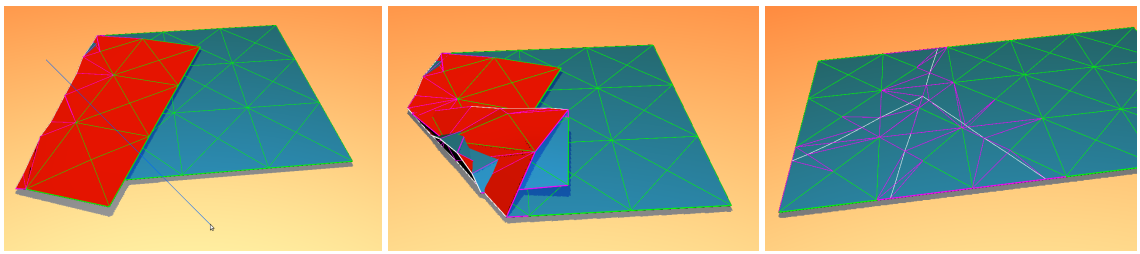
By altering the movement radius σ of the Gaussian, more centralized and more localized movements can be implemented (see Figure 6.4).
The new model, along with the generalized control law, can be used as an alternative to the original model. However, rather than using the new model to re-do the original experiments, its capabilities are now demonstrated by employing it in the following extensions for marker-less paper detection (see Sections 6.2 and 6.3) and automatic fold detection (see Section 6.4).

6.2 Kinect-based Paper Detection

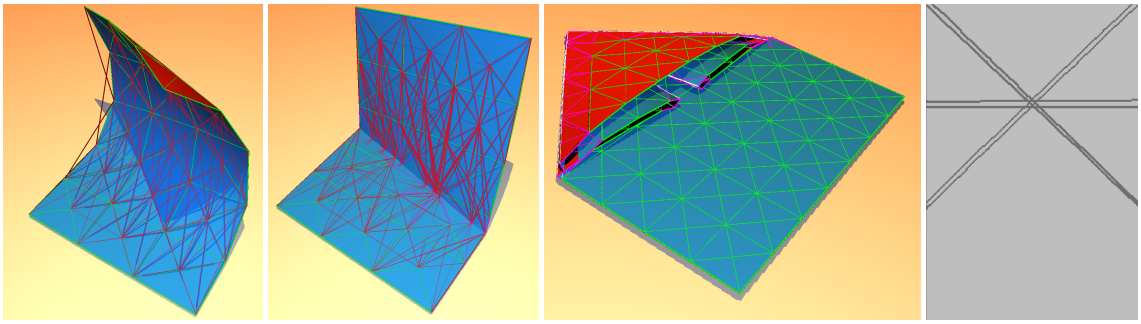
One of the main drawbacks of the paper tracking approach thus far (see Sections 4.2 and 5.2) is the necessity to cover the paper with markers. While the use of fiducial markers significantly facilitates tracking, it also severely decreases its *suitability for daily use*, since augmenting all paper with markers is not feasible. Due to the availability of affordable RGBD-cameras such as the Microsoft Kinect device, an obvious advancement would be to replace the current multi-view marker tracking with a new detection framework based on point cloud processing. However, before a prototype marker-less system is proposed, the particular issues solved by the marker-based detection approach are highlighted again. In the absence of 3D information, multi-view approaches must solve the registration issue in the different camera views, i.e. the system has to find identical points



(a) Fold through curved paper (b) After relaxation (c) Possible deformation

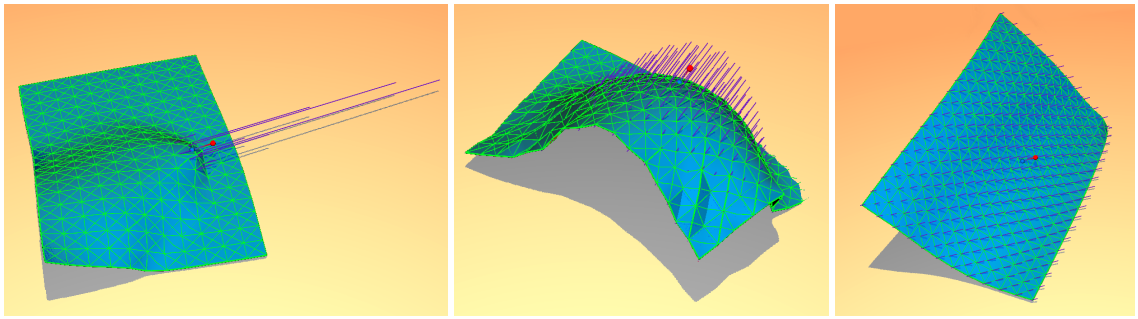


(d) Fold line through several layers (e) Successive folds (f) Resulting fold lines



(g) 2nd order links (h) No links along crease (i) Complex double quash fold (j) Fold-map

Figure 6.3: Demonstration of the presented mouse-based technique for manually adding fold lines. **(a-c)** Curved fold lines. **(d-f)** Intersecting fold lines. **(g-j)** Bending constraints and fold map. **(a)** The mouse gesture results in a straight (blue) line L, but the paper model is curved. All triangles of the current deformed paper model that intersect with L are split. **(b)** Once the fold is added, the paper can be transformed back into its undeformed state, revealing the resulting curved fold line. **(c)** The curved fold line does not allow the paper to be actually folded, but bending along the line is possible. **(d)** Another model was folded once and a second fold line is added. The new fold line intersects both layers of the paper model. **(e)** The second iterative fold was applied. *Note that the visible clipping artifacts are caused by an internal tension in the physical model that is caused by the physic engine's self collision handling. Deactivating self collisions suppresses these issues, but allows impossible folding operations to be performed.* **(f)** After unfolding the model, the resulting fold line structure is revealed. The second fold line was only straight, when it was applied to the deformed paper. **(g)** In its initial state, the paper nodes are connected by bending constraints if their pairwise distance on the paper surface is below a given threshold. **(h)** Once a fold line is added the intersecting links are deactivated by setting their stiffness to a very small value (10^{-5}). **(i)** The mouse based interaction framework even allows complex double squash folds to be added to the paper model. In order to lower the model tension in the vicinity of the sharply intersecting folds, each fold line was added twice. **(j)** Fold map of the model before performing the double squash fold. It shows that each fold line was added twice.



(a) Small σ (b) Medium σ (c) Large σ

Figure 6.4: Effect of different movement radii σ. In order to amplify the local movement of the paper for visualization, the global paper stiffness coefficient was set to 0.01, which made the paper behave *softer*, i.e. more like a tissue than like common printer paper. The lines show the movement direction and the qualitative movement strength of each of the nodes.

in several views. Since the extraction of dense depth-maps using common multi-view geometry methods is known to have severe issues with homogeneous image regions and with their real-time applicability, key-point based method are often preferred. These, in general, estimate a large set of key-points in each of the images. Subsequently, at each key-point position, a generic image feature vector is computed that usually describes the image's local color distribution and/or gradient information in the vicinity of the key-point. In the subsequent matching phase, the feature vectors of different views are pair-wise compared. The ratio between the matching quality of the best match and the second best match yields a good indicator for the overall specificity of the match, allowing good matches to be filtered out using a simple threshold. If a feature match in at least two views is found, the key-point's 3D position can be computed using standard triangulation methods.
However, Rublee et al. [2011] showed that existing state of the art generic image features[1] struggle with matching reliabilities of only between 30-50%. Actually in our experience even these numbers are only attainable with images showing either high contrast textures or structures. This means that an empty sheet of paper might offer few, if any, generic features that could be used to solve the registration issue in a multi-view system. Moreover, the manipulating hands that also *produce* features have to be taken into account.
In order to still use generic image features even in the presence of more outliers than inliers, the rigidity of a tracked object is commonly taken into account to allow a *consensus set* of inliers to be statistically filtered out using RANSAC based methods. However in the present case of deformable objects, there is no *rigidity* that can be exploited. For deformable objects that have a distinct number of links and joints, each link could be tracked separately by performing RASAC only on the key-points associated to that link. However, even assuming a characteristic texturing of the manipulated object, these methods can still not be transferred to compliant objects such as paper or cloth directly. The *continuous* deformability could be counteracted by approximating the object surface by a large number of small rigid patches – comparable to the physical modeling that we presented – but the smaller the patch size, the less likely it is to find a sufficient number of distinguishable features. The marker-based detection engine we implemented simply uses the defined key points provided by the markers to bypass all of these issues.
The Microsoft Kinect device circumvents the registration problem internally using a structured IR-light emitter that projects a – for the human eye invisible – quasi-random speckle pattern into the scene. From the displacement of the dots in the IR-image and the known stereo axis between the IR camera and the IR emitter, an 11bit depth image is computed in hardware on the camera de-

[1] They compared SURF, SIFT and their own ORB features

vice. Together with intrinsic and extrinsic camera parameters obtained from camera calibration[2], a non-linear transform is used to compute a metric depth image that is then again used to obtain a 3D point cloud. Unfortunately, the color camera in the Kinect is not pre-calibrated, necessitating an expensive mapping of the color image into the point-cloud in software to associate color information to point-cloud points.
However, while the use of Kinect for point-cloud acquisition indirectly avoids the multi-view registration problem, finding an association from a detected 3D point in the world to the corresponding 2D position on the paper surface, previously accomplished using fiducial markers printed onto the paper, cannot be solved in a trivial fashion. A simple estimate of this association can be calculated if temporal consistency is assumed. Once the paper-model and the point cloud of the paper are correctly aligned, the assumption of small paper movements between two consecutive processing frames allows us to assign the nearest point in the point cloud to a given model position, i.e. by performing Iterative Closest Point (ICP) [Besl and McKay, 1992] for tracking. We implemented an initial Kinect-based tracking prototype in order to investigate how well this approach can cope with configurations in which the assumption of temporal consistency is wrong, i.e. when the tracking of the paper is lost. The remaining question of how to segment the point cloud in order to filter out non-paper points is initially solved using a bright blue sheet of paper in front of a non-blue background, which allows simple color-based segmentation to be used.
In 2003, Haehnel et al. [2003] presented an extension of the ICP algorithm that allows deformable objects to be tracked. Due to the lack of a ready-to-use physics engine, they defined simple soft-links between neighboring model nodes and because of the absence of appropriate sensors, they did not address the real-time applicability of their system. Another comparable system for Kinect-based tracking of 3D objects was presented by Schulman et al. [2013b] with impressive results. However, while their system conceptually generalizes to using arbitrary point-cloud features, they used a completely homogeneous model that did not take into account folds. Instead, a low global model stiffness was combined with thick materials so that the created folds could satisfactorily be modeled by narrow bends. Furthermore, we note that the basis of our algorithm is significantly simpler than the EM-based algorithm they used. Most recently in 2017, Petit et al. [2017] presented a further comparable system to track the deformation of a pizza-shaped object. Based on state-of-the-art GrabCut segmentation and spatio-temporal tracking, they used a model based on finite elements methods that they tracked using an ICP-variant for deformable object.
While all of these system are conceptually very similar to our initial approach, non of them used a model that explicitly handles folds. In addition, we first present an extra set of heuristics to improve the tracking results (see Section 6.2.2). Subsequently, we include 2D-image-based SURF-feature tracking into our ICP pipeline, which leads to an astonishing improvement in tracking performance. (see Section 6.3). Finally, even methods to automatically detect when and where folds are added to the model are presented and discussed (see Section 6.4).

6.2.1 A Kinect-based Prototype for Tracking Paper

In this and in the following sections, a prototype system for marker-less paper tracking is presented. Starting with a straight forward ICP-based approach, the system is enhanced heuristically in a step-by-step manner in order to deal with the most obvious issues. We emphasize that our goal was not to completely*solve* the issue of marker-less tracking of a sheet of paper being manipulated by a human in this work. Rather, the presented and evaluated iterations of the system should be considered as a feasibility study towards such a system. We employ the generalized model presented in Section 6.1.
The processing pipeline (see Figure 6.5), can coarsely be subdivided into three processing steps.

[2] This is also natively supported by ICL (see Figure 3.5)

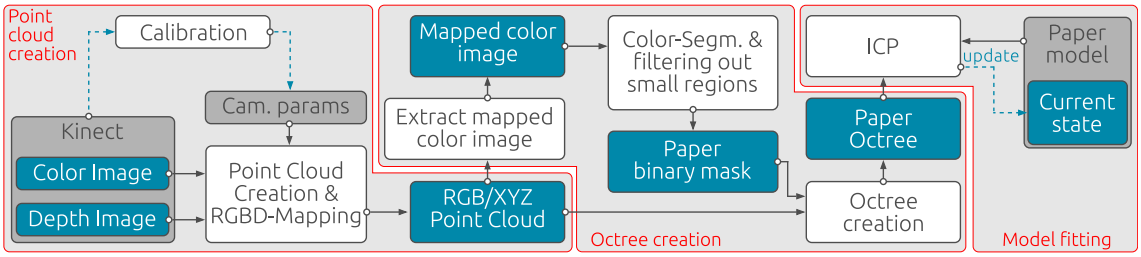
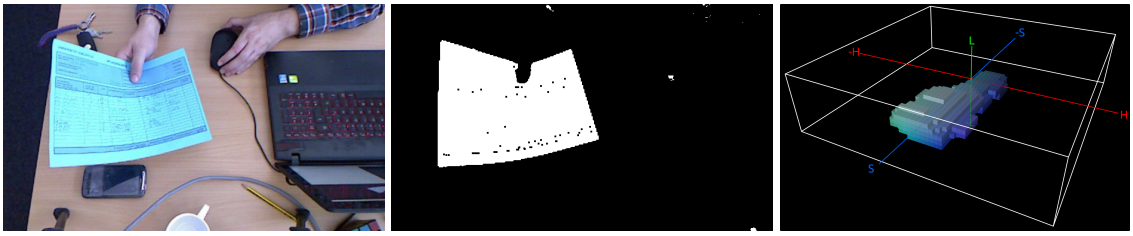


Figure 6.5: Processing pipeline for the initial kinect-based paper tracking framework. The whole pipeline can be coarsely divided into the three parts *Point cloud creation*, *Octree creation* and *Model fitting*



(a) Input color image (b) Eroded binary image (c) 3D Segmentation LUT

Figure 6.6: Color segmentation. **(a)** Input color image. The segmentation task is rather simple because the paper color can be easily distinguished. **(b)** Resulting binary mask. In order to remove single pixel outliers that often occur due to bayer-decoding artifacts along image edges, the segmentation output is post-processed with an erosion filter. Remaining small foreground regions, such as the one corresponding to the Intel logo on the notebook, are filtered out by using only image regions that have at least a chosen minimum amount of pixels. **(c)** Visualization of the 3D color distribution of the paper. Due to the use of a different amount of significant bits per channel, the distribution space is flat. A common practice is to use less bits for the lightness channel to better cope with changing lighting conditions and shadows. In the example, 6 bits were used for the hue and the saturation channel, but only 4 bits for the lightness channel.

In the first step, the scene's point cloud is created. To this end, the calibrated camera parameters are used to create the points' XYZ coordinates. In parallel, the relative transformation between the Kinect's color camera and the Kinect's depth camera are used to map the provided RGB image into the point cloud, resulting in an additional mapped RGB-tuple for each point.
Subsequently, points that do not belong to the paper have to be filtered out. After extracting a color image from the point cloud, which is necessary to obtain the image data in a supported planar pixel layout, color segmentation is used to create a binary paper-pixel mask. The employed color segmentation method uses a certain number of significant bits per HLS color channel for efficient indexing in a 3D look-up-table, which represents the color distribution of the paper. An example is shown in Figure 6.6. For additional details about the segmentation method, which is also available in ICL (see Section 3.3), the reader is referred to Schroder et al. [2012]. By using the mapped color image for the color segmentation, the binary mask's x/y-pixel positions directly correlates to the x/y-positions of the points of the organized input point cloud. In the initial ICP-based model tracking approach, for each model node, the closest paper point cloud point must be found. This crucial step is performed in each ICP-iteration, so when using a naive nearest neighbor search, the overall complexity is $\mathcal{O}(N \cdot M \cdot K)$, where N is the number of model nodes, M is the number of paper point-cloud points and K is the number of ICP-iterations. For the case of standard rigid body ICP the update steps between ICP-iterations are commonly performed directly by computing a single affine mapping between the current points and associated points, which usually results in a fast convergence after less then five iterations. In contrast to this, in our method, the affine

	Creation	NN search	Approx. NN search
PCL Octree [1mm resolution]	19ms	1000ms	250ms
PCL Octree [10mm resolution]	8ms	1400ms	185ms
PCL Octree [32mm resolution]	4ms	2700ms	300ms
ICL Octree [Leaf capacity 16]	6ms	45ms	11ms

Figure 6.7: Benchmark of the two Octree implementations. For the creation, a full 320×240 point cloud was inserted into the Octrees. Subsequently, each contained point was used for a nearest neighbor search and for an approximate nearest neighbor search. A further decrease of the PCL Octree resolution did not provide better results.

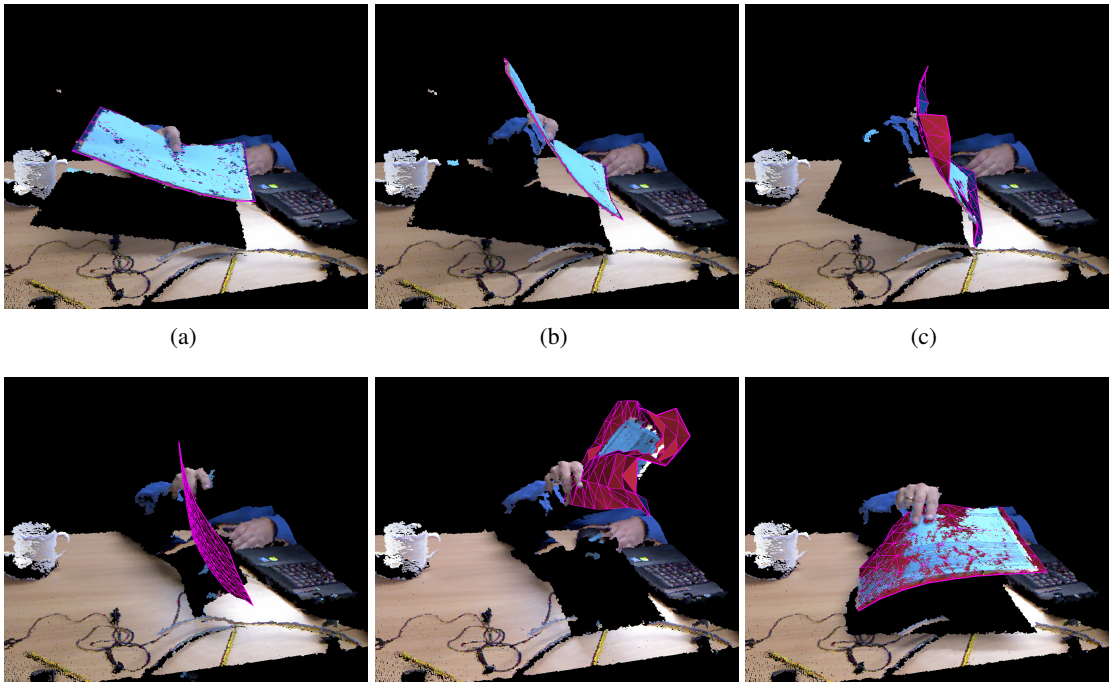
rigid body mapping is replaced by applying the P-controller-based model control law (see Section 6.1) for each association, which works, due to the coupling with the physics engine, less directly. This means that our ICP tracking implementation requires more iteration cycles to reliably reach convergence.

A common method to reduce the ICP complexity to $\mathcal{O}(N \cdot \log M \cdot K)$ is to pre-organize the point cloud data in a hierarchical spatial search structure, such as an Octree or a KD-Tree. In a first attempt, the PCL-Octree implementation [3] (see Section 3.2.6) was integrated into the system. However, it turned out that the provided nearest neighbor search mechanism was at least an order of magnitude too slow for the desired real-time performance of the system. As a replacement, an Octree module was implemented and integrated into ICL. The implementation was realized as a generic class template that employs several compile-time constants for optimization of data-handling, fixed-point-approximation and memory allocation. In contrast to the PCL-Octree implementation that operates on top of an existing point cloud and deals with point indices, the ICL-implementation directly contains the point cloud points in order to optimize memory accesses internally. While the time for the Octree creation is comparable in both implementations, our implementation's nearest neighbor search is at least 20 times faster. A concise benchmark comparison is given in Figure 6.7. The PCL Octree is defined by an internal leaf node resolution. Each cubic leaf node contains a list of points that are in the corresponding voxel. In contrast to this, the ICL Octree implementation uses a maximum capacity for each leaf node. In general, the use of *larger* leaf nodes make the Octree creation faster, but the nearest neighbor search slower. It is important to mention that the PCL Octree only supports a more general K-nearest neighbor search which was called with $K = 1$ for the benchmark. For the ICP-based tracking, the points of the paper segments of the point cloud are inserted into the Octree structure. This can be done in negligible time since the number of points that are actually inserted is only a fraction of the size of the whole point cloud. Subsequently the ICP-loop is entered and iterated for a fixed number of cycles.

While the basic formulation of the ICP algorithm is used to align two rigid 3D point sets, in our case, only the observation data is given as a 3D point set. However the grid-nodes of the model provide a well suited approximation of the model surface and thus are analogous to a 3D point cloud. Therefore, per iteration step, each model node is moved into the direction of its nearest neighbor in the paper point cloud, if the distance to this nearest neighbor is not larger than a threshold that was manually set to 50mm. The model movement (see Section 6.1.3) is realized with a very small movement radius so only the actual node is affected by its movement. Using 10 ICP iterations, a sufficient tracking adaptability of the model is reached. Even though a very visible tracking latency exists, the model usually never gets lost. However, due to the small computational footprint of each cycle, it takes more then 90 ICP iterations before the overall tracking frame-rate is affected[4]. We note that the Kinect acquisition frame-rate is itself limited to 30 Hz. At

[3] `pcl::octree::OctreePointCloudSearch<pcl::PointXYZ>`

[4] On an Intel® Core™ i7-4700MQ CPU @2.40GHz, 16GB Ram, 64Bit Ubuntu Linux



(a) (b) (c)

(d) (e) (f)

Figure 6.8: Tracking the paper while it is turned upside down. When the paper is too parallel to the camera's view-axis, it becomes invisible for the used Kinect device. **(a-c)** The paper normal is rotated away from the camera's view axis direction. **(d)** The paper is lost by the Kinect camera and the model is slowly straightened by the physics constraints. As the model tracking is only applied when the paper point-cloud is visible, the model orientation is not adapted. **(e)** Once one part of the paper point cloud becomes visible again, tracking continues. However, since only the top-part of the paper is visible, also the bottom part of the model is drawn upwards leading to a curled model surface. **(f)** Only when the whole paper point cloud becomes visible again, the model straightens.

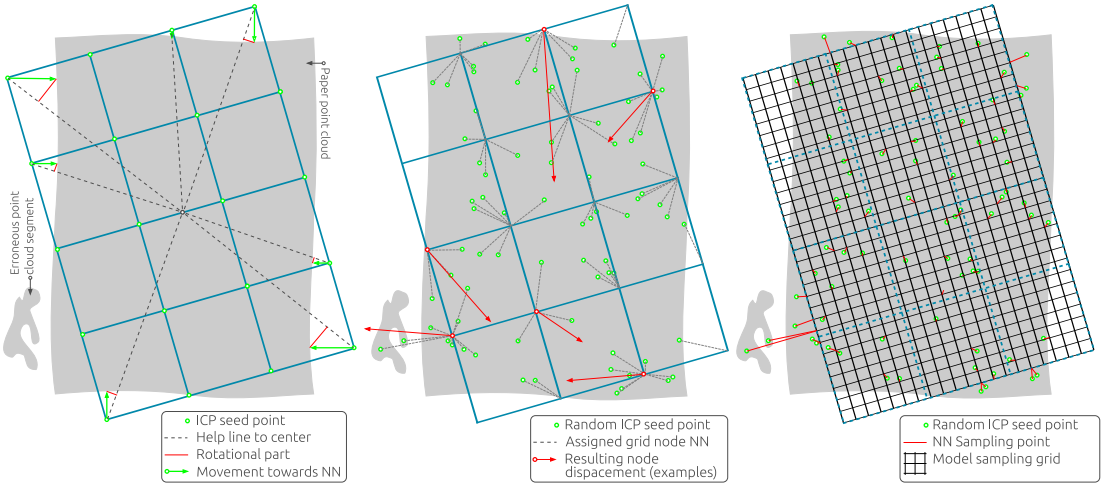
approximately 50 iteration cycles, the tracking system seems to be saturated so that more iterations do not lead to an additional obvious improvement.

6.2.2 Strengths, Weaknesses and Heuristical Improvements

Despite our rather simple approach, the tracking performance of system is very impressive as long as the paper deformation is moderate and the paper is mostly oriented towards the camera. However, in contrast to the marker-based tracking system, the new system can not distinguish which side of the paper is oriented towards the camera or one long/short edge of the paper from the other one, which means that whenever the system recovers from a loss in tracking of the paper deformation or orientation, the model might have flipped its sides. Whether this has undesirable consequences or not, depends on the actual application.

A more severe issue of the kinect-based tracking system emerges if the paper's surface normal direction gets orthogonal to the direction of the camera's view axis (see Figure 6.8d). If this happens, the Kinect cannot detect the paper surface and the paper simply disappears from the scene. As long as the paper remains in this relative orientation to the camera, the nearest-neighbor-based tracking does not detect any close neighbors and the tracking is disabled. Figure 6.8 shows a very common paper movement scenario in which a sheet of paper is turned over. Due to the fact that this issue actually emerges from the image acquisition in the camera, there is no straight

forward way to fix it. A possible solution would be to use two Kinect devices with orthogonal view axes, but in that case the surface re-construction quality deteriorates due to the interference of the two speckle patterns.
Figure 6.8e highlights another major issue of this tracking approach. In cases in which only a part of the point cloud is visible, the model can become curled as more and more nodes are drawn into the *attractor field* of the visible section of the paper. A workaround for this issue is to invert the ICP-tracking, i.e. to not find a nearest point cloud neighbor for each of the model nodes, but to find a nearest model node for each (or for a subset) of the paper point cloud points. Since the point cloud has a variable size that is usually at least one or two orders of magnitude larger than the number of paper nodes, only a subset of the point cloud points can be used as origins for the nearest neighbor search if real-time applicability is to be preserved. In order to achieve a better control of the size of the subset, the number of random samples that is drawn from the paper point cloud can be linked to the current number of paper nodes. Using four times as many random points than model node points provided a good trade-off between speed and ensuring with a high probability that the majority of the model nodes are selected at least once as a nearest neighbor. This adaption of the ICP heuristic fixes the issue of a curling of the paper model if only a part of the paper point cloud is actually visible. However, it also introduces two new issues. While the original version was not responsive to segmentation errors as long as the paper remained visible (see Figure 6.9a), the adapted version tends to pull the model towards erroneously detected paper point cloud segments (see Figure 6.9b). In addition, the coarse sampling of the model leads to a strong randomization of the movement targets for single nodes, which makes the model shaky and the physics engine slower. Increasing the number of samples that are drawn enables a stronger balancing of these updates, but it also increases the resulting magnitude orthogonal to the model surface, which can lead to alternating model movements due to an overshooting of the target position. This, in turn, must be counteracted by decreasing the movement amplifiers. While the affinity to attract the model towards erroneous point cloud segments is a general issue of the inverted tracking method, the randomization can be decreased by not only using the model grid nodes, but by further sampling the model surface in order to get a more detailed approximation. Alternatively, it would have been possible to extend the ICP implementation internally, to estimate closest points by minimizing point-to-triangle distances, but while a sketch of this extension was carried out, implementation was not feasible given the limited remaining time. Moreover, it is not clear that the results would actually be better as one can easily argue that the two methods converge as the model sampling resolution is increased. Figure6.9c shows the decrease of the node-movement randomization. However, the sampling-heuristic comes with a severe increase of the computational cost for each ICP step, which then does not only consist of nearest neighbor searches and applying the physics engine, but also of sampling the model and inserting the model points into an extra Octree to be able to perform the nearest neighbor search efficiently. This limits either the number of ICP cycles that can be performed at full frame-rate (here, only about 10 rather than 90 cycles are possible) or the tracking frame-rate itself, which in turn makes the tracking more difficult due to larger frame-to-frame movements of the paper.
Another issue that can be observed with both ICP methods is a deterioration in tracking speed for in-plane model movements, in particular in-plane rotation. A possible reason for this is the small tangential movement force component (see Figure 6.9a) of the velocity-based control. While for movements along the surface-normals each movement contributes mostly to move the model to the target position, in the rotation case, only the corner nodes and a few adjacent edge nodes actually provide a rotational impulse. The same effect exists for the case of translating the paper in-plane, because here, also only the edge nodes that are *moved out of* the point cloud contribute to the movement. A first attempt to correct this involved amplifying the control velocity in cases where the node movement direction is more orthogonal to the nodes surface normal $\hat{\mathbf{n}}_i$, which leads to a



(a) Original version (b) Inverted ICP heuristic (c) Inverted with sampling

Figure 6.9: Analysis of the model updates in case of in-plane model movement for the different heuristics. The images assume the model (blue grid) and the point-cloud (light gray background) to be perfectly aligned in the z-direction. **(a)** The initial ICP method provides small rotational updates only for the corner nodes and for a few adjacent edge nodes. Erroneous paper point-cloud segments usually do not have a negative influence on the tracking performance unless they are close to the model. **(b)** The inverted ICP heuristic avoids *curling* of the paper if only a part of the whole paper point cloud is visible (not visible in the figure), but it has issues caused by a stronger randomization of node movements and with erroneously detected paper point-cloud segments. **(c)** Sampling the model in order to achieve a better point-cloud approximation minimizes node movement randomization, but also incurs a significant reduction in speed.

refinement of Equation 6.2:

$$\mathbf{v_i} = \begin{cases} \alpha_i \lambda \mathbf{d}(1 + \beta \hat{\mathbf{n}}_i^T \hat{\mathbf{d}}) & \alpha_i > \alpha_{\text{min}} \\ \mathbf{0} & \text{else} \end{cases}, \tag{6.3}$$

where $\hat{\mathbf{d}}$ defines the normalized displacement direction and β defines the gain of the compensation [5]. Before the movement law (see Equation 6.3) was finalized avoiding movement amplifications that are not orthogonal to the node's surface normals, an experiment was carried out in order to test whether the compensation term actually improves the tracking of in-plane rotations. Using a synthetically created rotating paper point cloud as input, the experiment showed that the fix only has a minimal effect. By repeating the same rotation tracking experiment with different model grid resolutions, it turned out that the tracking becomes worse with larger numbers of model nodes, leading us to the conclusion that the rotational velocities, applied to the corner and edge nodes are mostly compensated by the resting majority of inner nodes. A strategy to handle this issue would need to also suggest movement directions for the inner model nodes. This idea is readopted for the extension of the point-cloud based tracking system presented in Section 6.3. Here, inner model updates are generated by detecting SURF-features in the Kinect color image that mapped into the point-cloud.

6.2.3 Folding the Paper in Half

It was important to test the capabilities of the Kinect-based tracking system during an interesting interaction Scenario. However, unlike the marker-based tracking system, which worked for

[5] i.e. $\beta = 0$: no compensation, $\beta = 1$: maximum compensation factor is 2, etc.

many complex interaction sequences (see Section 5.4.2), here, we restricted the experiment to a single folding action (see Figure 6.10). To be able to track more complex interactions additional extensions are needed. The figure shows an unsuccessful (see Figure 6.10a-d) and a successful tracking trail (see Figure 6.10e-i) and once again underlines the lack of a reliable association between point-cloud points and model surface coordinates. The tracking of the manipulation fails when the bent part of the paper is lost by the Kinect due to its orientation towards the camera (see Figure 6.10c). When this occurs, all parts of the model are drawn towards the remaining visible point cloud, leading to an unfolding of the already modeled 90 degree fold. When the bent part of the paper becomes visible again the tracking mechanism cannot recover as the discrepancy between the model and the point cloud data is too large.
In a second exemplary sequence (see Figure 6.10e-i), the movement was carried out more quickly which reduced the time when the top part of the paper was invisible and thus the model does not have time to erroneously unfold completely and therefore the tracking continues correctly. However, once the model reflects the folded state of the paper, the tracking technique does not suffice to track subsequent unfolding (see Figure 6.10j-l). It is important to mention that the success rate for tracking the folding operation was less than 10% and it strongly depended on the relative orientation of the paper towards the camera, the overall speed of the manipulation and thus the movement speed of the paper parts and the amount of occlusion. In particular, the fact that in its current form the system cannot recover once tracking is lost means that it has limited value. Attempts to track more complex scenarios were not carried out given these failings.

6.2.4 Discussion

The development of the prototype system for point-cloud based detection of a manipulated sheet of paper helped to get valuable insights about the potential and the difficulties of such an approach. While the initial ICP-based algorithm provided promising results, the introduction of heuristics to fix some of the most obvious issues led to new errors. However, these prototyping studies allowed us to illuminate the two main issues of the Kinect tracking system.
The first and most severe drawback is the lack of a reliable association mechanism between detected paper point-cloud points and 2D model coordinates. The only mechanism available to us is the spatio-temporal one in which the previous state of the model is used to drive the current state. This is sufficient if the paper remains fully visible *and* the movement is slow. An obvious first improvement would be to try to identify edges and corners of the paper in the point-cloud, but the development of an additional prototype led to severe complications when confronted with occlusions and deformations. Therefore, a more complex but also more general extension of the original tracking approach was implemented that enhances the ICP-based tracking using SURF-features[6] that are detected in the Kinect's color image and then mapped onto the model surface. The final tracking system, presented in Section 6.3, then employs a combination of ICP-based and SURF-feature-based tracking.
The second drawback, which turns out to be even more difficult to overcome, is that the method only *pulls* the model towards detected paper point-cloud points, it does not *push* it away from positions where no points were found. Solving this requires us to overcome two major difficulties. First, the system would have to be able to distinguish whether the *non-existence* of a paper point-cloud patch around a given position allows to infer that the real paper is actually not there. Alternatively, the paper could just be invisible due to (self-)occlusion or due to the perception geometry itself. A proper differentiation of these situations would require the system to actually model the perception system, in a similar way to the view-based model-tracking approach of

[6] The actual feature type must not necessarily be SURF; the implemented system only requires features to have a 2D image anchor.

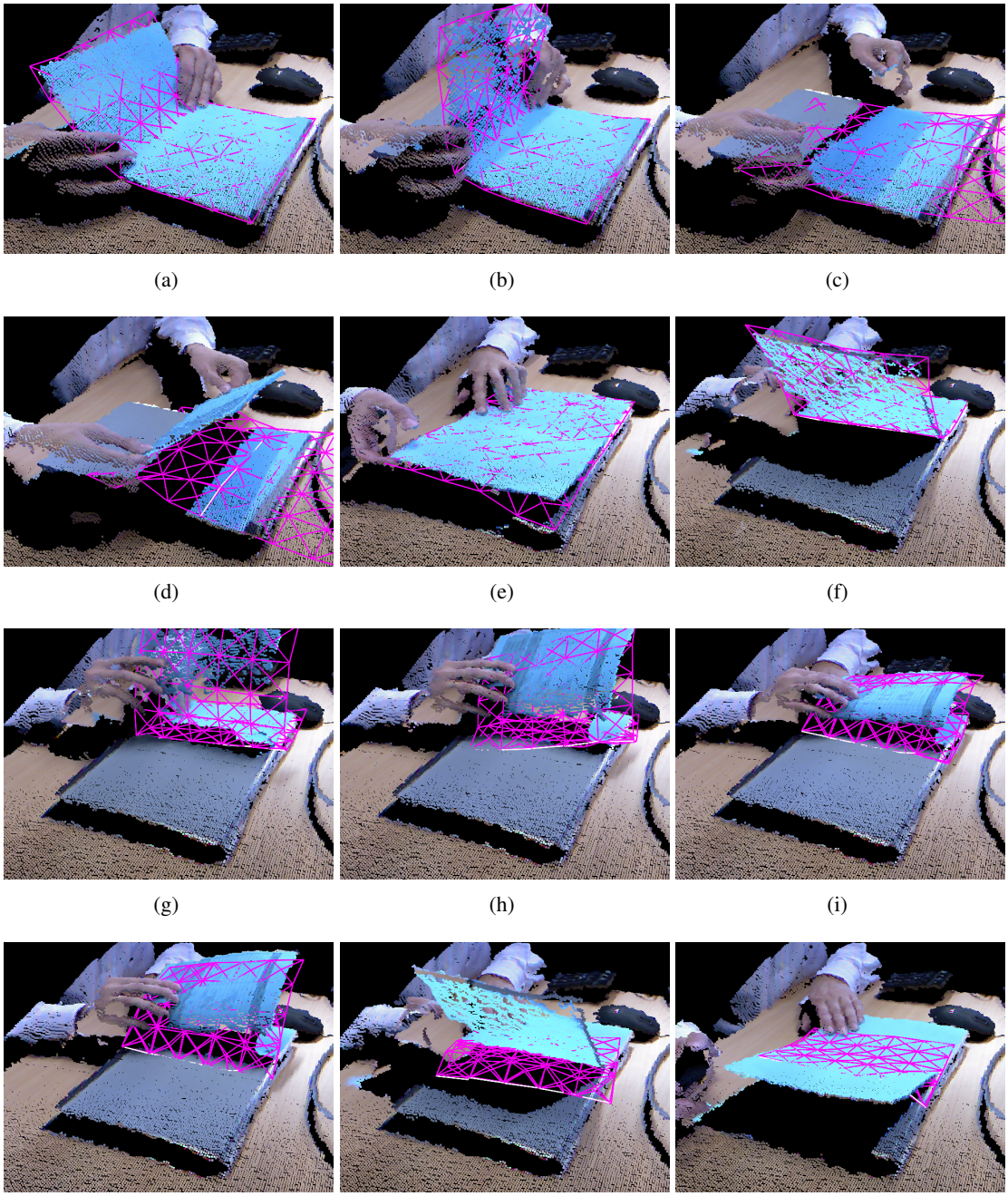


Figure 6.10: Kinect-based tracking of a sheet of paper while it is folded in half using the Kinect-based tracking system. We here used the initial version as the inverted ICP heuristic yielded worse results. A central fold line was manually added to the model beforehand. **(a-d)** In the first sequence the motion was slow and the tracking failed. As soon as the grasped part of the paper is oriented towards the kinect device it disappears and the model is dragged towards the visible part of the paper point cloud. When the missing part of the paper reappears, the tracking cannot recover (d). **(e-i)** In the second sequence that was explicitly performed with a faster movement, the folding operation could actually be tracked successfully. Due to the faster movement, the model did not relax its deformation in the short time where the bent part of the paper was not visible (between f and g). After this, the tracking recovers, so the final folded state of the paper is recognized (i). **(j-l)** However, a following unfolding of the two parts is not recognized by the tracking system as the two folded parts are not explicitly distinguished.

Oikonomidis et al. [2011]. Secondly, a repulsion mechanism is needed to make the model avoid locations where the actual paper is known not to be.
Due the immense programming effort required for such an approach, which would only attain real-time performance if major parts of the system were ported to a GPU-based implementation, this was not pursued any further. Furthermore, the manipulating hands would also have to be tracked and modeled in real-time in order to correctly model occlusions.

6.3 Supplementing Point-clouds with 2D-SURF-Features

In the previous ICP-based approach, associations between 3D point cloud patches and 2D model coordinates were derived using only spatio-temporal information, which led to severe issues in cases where the tracking was lost. In order to obtain an additional information source to augment this association mechanism, a new approach that extends the original ICP-based tracking by attaching SURF-features to the paper model surface, was implemented.
While the original ICP-based tracking was not adapted, the new SURF-feature-based mechanism is triggered concurrently and yields additional model node updates for coarser paper movements. Several adaptions to the system structure and also to the paper itself were necessary to facilitate this update to the system.
As our method can employ any kind of 2D image features, we omit a detailed investigation of SURF-features. We decided to use SURF-features because of their known combination of accuracy and speed and therefore integrated a SURF-feature tracking module into ICL (see Chapter 3) that provides a unified interface for a CPU and a GPU-based detection backend. In order to combine the option for trivial color-based segmentation with reliable SURF-feature detection, a blue-tinted graffiti image was printed on both sides of the paper (see Figure 6.12a).
After describing how the original ICP-based processing pipeline was extended (see Section 6.3.1), the results of a set of new interaction experiments are presented (see Section 6.3.2) and the generalization of our approach when tracking the folding of commonly textured paper is evaluated (see Section 6.3.3). Finally in Section 6.3.4, we discussion the results of our marker-less tracking approach in general.

6.3.1 Extending the ICP-Pipeline by SURF-feature Detection

The input to the SURF-feature detection pipeline is a gray-scale converted instance of the original pipeline's mapped RGB color image, which is as an intermediate processing result that was already available (see Figure 6.11). However, the image cannot directly be used for SURF-feature detection. A number of initial experiments showed that the low texture contrast, emerging from the artificial necessity of a *segmentation-friendly* blue-in-blue texture, leads to a very low number of significant SURF-features. Therefore, a contrast enhancement operation must be applied to the gray-scale image. To this end, a custom high speed local contrast operator was employed.

Local Contrast Enhancement for Optimized SURF-Feature Detection

State-of-the-art methods for contrast enhancement usually aim to reach an equalization (or a simpler stretching) of locally computed histograms and are therefore computationally very expensive [Kokufuta and Maruyama, 2009]. In contrast, the proposed method is based on a region's average gray-value only, which can be computed in constant time when exploiting the properties of the integral image representation introduced by Viola and Jones [2001]. The correction of an input

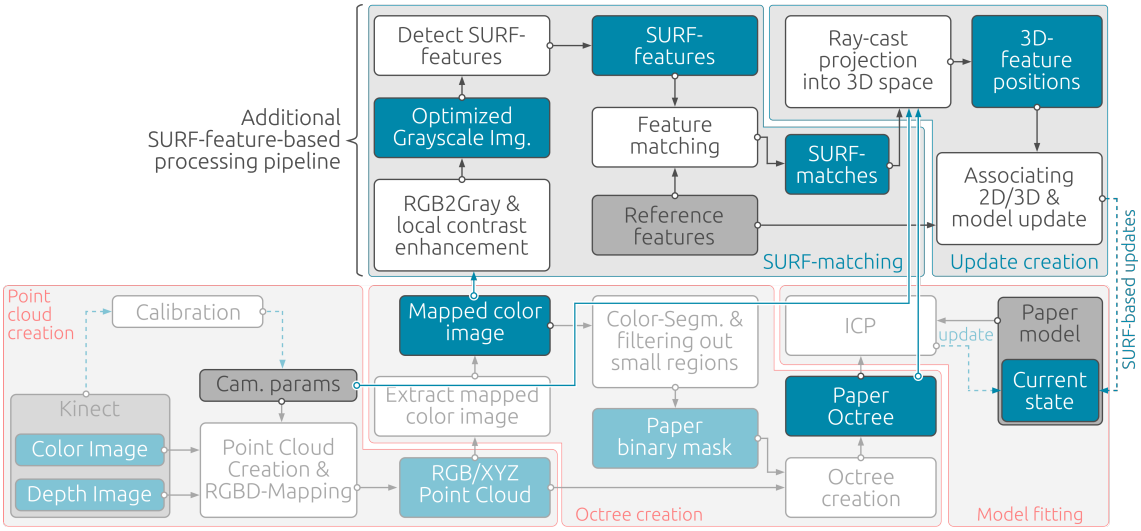


Figure 6.11: Extended processing pipeline including the original ICP-based model tracking (bottom, red boundary, for more details refer to Figure 6.5) and the new additional SURF-feature-based tracking components (top, blue boundary). The new pipeline is subdivided into a *SURF-matching* part and an *Update creation* part. While the ICP-based pipeline remains independent from the new components, the SURF-feature-based pipeline uses several existing intermediate processing results from the ICP-based pipeline, such as the RGB-D-mapped color image and the paper-Octree.

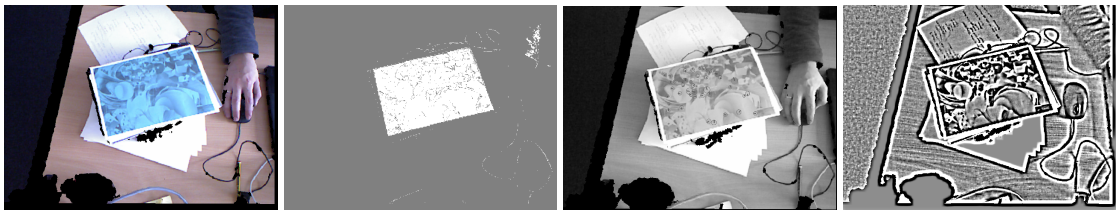
pixel's intensity value g_{xy} is computed using the following linear function

$$f(g_{xy}) = s \cdot (g_{xy} - \mu_{xy,wh} + \theta_{\text{global}}) + 127, \tag{6.4}$$

where s is the linear slope, $\mu_{xy,wh}$ is the average gray value in the rectangular region of size $w \times h$ centered at (x,y) and θ_{global} is an additional global threshold. The results are clipped to the range [0,255]. The open parameters of the operator, the global threshold θ_{global}, the neighborhood size $w \times h$ and the slope s allow a wide variety of results to be obtained. We found the most crucial factor to be the mask size. While a very small mask size, e.g. $w = h = 3$, yields a Laplacian-like border image, a very large mask size approximately a third the size of the input image size, results in a local contrast equalization. An intermediate mask size, 10×10 pixels, yields a good combination of both (see Figure 6.12d) leading to a strong amplification of local contrasts. Depending on the used slope value, gray-scale gradients can be also be intensified. A list of the important intermediate image processing result images is presented in Figure 6.12.
Figure 6.13 visualizes the SURF-feature matching performance that was reached by employing the presented local contrast enhancement method. Even in the presence of a cluttered background and despite the fact that the front face texture and the back face texture look – at least to the human eye – very similar, an outstanding matching performance is achieved. In the depicted snapshot (see Figure 6.13b) more than 50 real matches but only 3 false ones were detected. Of course, it has to be admitted that the error ratio gets significantly larger in the presence of severe occlusions and in situations in which the paper is oriented further away from the camera, but in the running system, a very simple selection heuristic allows us to achieve good tracking results (see Section 6.3.1).

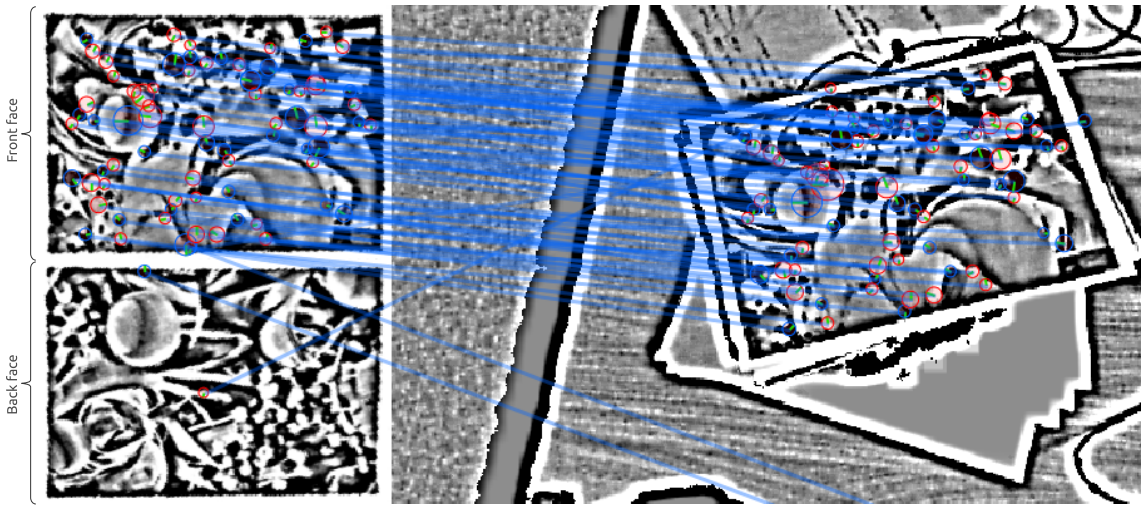
Octree-based Ray-Casting for Fast 2D/3D Association

An additional implementational difficulty was the fact that the mapping from the color image into the point cloud could not simply be created using a constant-time projective transform. Instead, an Octree-based ray-cast mechanism was implemented that uses a feature's view-ray to find the



(a) Input RGB image (b) Segmentation (c) Grayscale image (d) Local contrast enhanced

Figure 6.12: Intermediate image processing results **(a)** Input RGB image, derived from the Kinect point-cloud. In contrast to the raw color image, provide natively by the Kinect driver, the image is mapped into Kinect's depth camera's frame resulting in some black gaps, where no point-cloud points were detected. **(b)** Color segmentation result. Even though the paper is no longer uniformly colored with a single blue tone, the segmentation works satisfactorily well. The remaining single-pixel error regions are intrinsically removed by only using larger connected regions as the input for the paper point-cloud. **(c)** Converted gray-scale version of the input image (a) underlines the low contrast of the paper texture. **(d)** After local contrast enhancement, the paper region's texture uses the full scale of possible pixel values from black to white, allowing for a much better and more reliable detection of SURF features.



(a) Paper texture maps (b) Detected paper

Figure 6.13: SURF-feature matches between the two paper reference images (front and back face, left) and the current observation (right). Even though, the textures of the front and back face look very similar, there is only a negligible amount of mismatches in which features from the actual (front) side of the paper are erroneously matched to features on the back face's reference image. Without using the presented contrast enhancement method, only about 5 correct feature-matches were detected in the present situation.

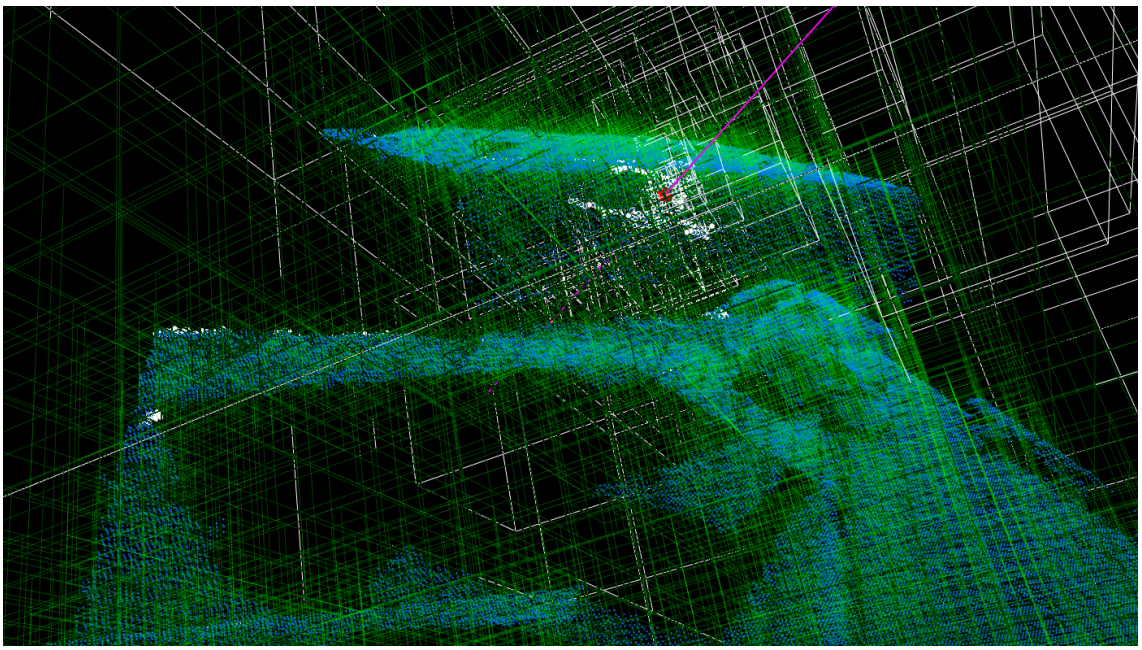


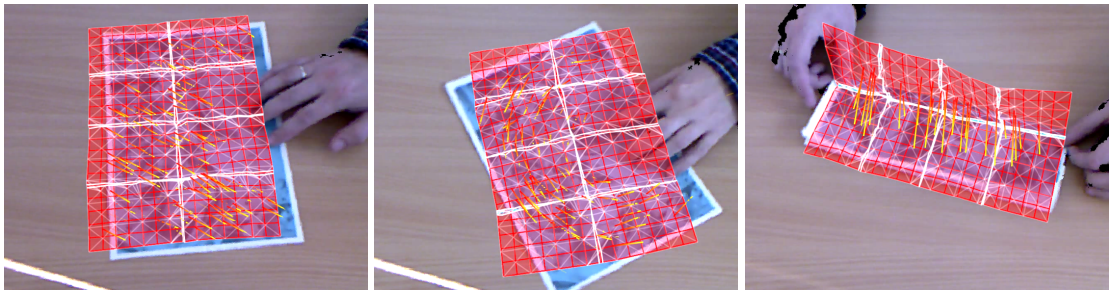
Figure 6.14: Visualization of the implemented Octree structure and ray-cast mechanism. In order to speed up the search of points that are within an epsilon-tube around the view-ray (purple line), only sub-volumes of the Octree are searched whose bounding sphere is close enough to the view-ray. Searched sub-volumes and points, actually used for the view-ray distance calculation, are highlighted in white. The resulting points, closest to the view-ray are highlighted in red.

corresponding point-cloud point closest to the ray (see Figure 6.14). The ray-cast mechanism implements a broad-phase search mechanism that pre-filters the point-cloud's points on the basis of the Octree's axis aligned bounding boxes (AABBs). Thus, only point-cloud points, whose parent AABB intersects with an epsilon-*tube* around the view-ray need to be processed. The use of an epsilon-tube rather simply the view-ray, is necessary to avoid missing very close points. If several point-cloud points are very close to the view-ray, the point closest to the camera is used. Here, the hierarchical structure of the Octree allows the ray-cast to be sped up significantly as for AABBs that are filtered, none of the child-AABBs must be processed at all. In the example in Figure 6.14, about 67000 points [7] are contained in the point cloud but the distance to the view-ray has to be calculated for only for 870 of them. For a speed comparison against the naive *point-by-point* comparison, a small experiment was conducted. 100 naive ray-casts take about 30ms for QVGA resolution and, due to the linear dependency to the number of points, about 120ms for a VGA point cloud. The Octree-based ray-cast mechanism performs the task in about 0.8ms and does not even take measurably longer in the VGA case, i.e. it provides a speed-up of factor 35 to 150, dependent on the point cloud resolution. The additional time needed to create the Octree can be neglected as the Octree is already available from a previous processing step.

Creation and Selection of Optimal SURF-Feature-based Model Updates

The SURF-features are detected in the grayscale converted instance of the Kinect's mapped color image. However, in order to use a detected SURF-feature, its current 3D world position and corresponding 2D paper-space position must be computed. For the latter, the fact that the obtained

[7] In the example, the point cloud is 2D-organized and has QVGA (320×240) resolution, but the unknown 3D points that correspond to the gaps in the depth image of the used Kinect-Device are heuristically discarded before the insertion into the Octree



(a) While shifting the paper (b) While rotating the paper (c) While folding the paper

Figure 6.15: Visualization of the resulting SURF-feature-based model updates. Each gradient (yellow to red) line indicates a matched SURF-feature and thus visualizes directly the associated model update, by connecting the position where a particular feature was detected in the world (yellow end) and the current position of the associated paper patch (red end). For the examples, the feature-match filtering technique of using only 90% of the actual feature matches (filtering out the longest updates) was employed. The thick white lines are the explicit fold lines that were explicitly added to the model.

feature association is not necessarily correct has to be taken into account. Initially, each SURF-feature match $M_j = (\mathbf{m}_j^m, \mathbf{m}_j^i)$ is described by two 2D positions, where $\mathbf{m}_j^m$ defines the position of the match in the template, i.e. in the paper-model space [8], and $\mathbf{m}_j^i$ defines the associated pixel position in the current input image. By using the Octree-based ray-cast mechanism with the view-ray that corresponds to $\mathbf{m}_j^i$, the 3D world position $\mathbf{m}_j^w$ of the feature is estimated leading to a simple model update: *move paper model position* $\mathbf{m}_j^m$ *towards* $\mathbf{m}_j^w$ *in the world*, which can be realized directly with the model movement law presented in Section 6.1.3.
However, due to the missing rigidity of the tracked object, erroneous feature matches cannot be filtered out statistically by means of RANSAC-based methods, another filtering mechanism is needed. While single outliers that would move a model patch to an arbitrary wrong position in the world is implicitly coped with very well by the constraints of the physical model, more frequently occuring outliers are likely to negatively influence the whole tracking performance.
It can be assumed that correct matches are statistically much more likely to produce updates with a smaller displacement $\|p(\mathbf{m}_j^m) - \mathbf{m}_j^w\|$ between the to-be-moved paper position and the target position than wrong matches that erroneously link two more or less random positions. Thus an almost trivial filtering heuristic can be defined. In each tracking step all model updates are sorted by their displacement and only the shortest 90% are actually executed. In cases where more errors are to be expected, such as when not using SURF-Feature-friendly textures (see Section 6.3.3), the percentage can of course be lowered. The heuristic implicitly provides good adaption properties in cases of larger paper movements. The relative contribution of erroneous matches linearly decreases with the magnitude of the actual movement of the paper. We expect erroneous matches to yield updates that are uniformly distributed but have on average a magnitude of approximately half the size of the field of view. Thus, in cases of larger movements of the paper, the likelihood to accidentally include erroneous updates increases. However, as in this case the relative contribution of the wrong updates with respect to the sum of the correct ones becomes increasingly negligible, the update mechanism is still able to handle the situation satisfactorily. As soon as the paper movement stops, the 90% cutoff works perfectly well again. Examples for the resulting set of SURF-feature-based model updates are given in Figure 6.15 and they are also visualized in the evaluation images in Section 6.3.2.
An advantage of the presented approach is the seamless combination of the ICP-based tracking

[8] For the actual internal implementation, the features on the front face of the paper and on the back face of the paper must of course be handled appropriately.

and the SURF-feature-based tracking. The ICP-based tracking is particularly well suited to fine-tune the model-surface locally, which can, due to the dense point-cloud-based target, also react to very small paper-patches. In contrast, the SURF-feature based tracking produces a much sparser and unevenly distributed set of updates that are, however, able to associate points independently from their distance in the world-frame. In contrast to the ICP-based tracking, a rather large image patch must be visible to robustly detect a SURF-feature.
The tracking system runs at frame-rates between 5Hz and 20Hz on an Intel®Xeon®E5-1620 running at 3.60GHz equipped with an NVidia®GeForce 660Ti graphics card. Even when using a full VGA XYZRGB-point cloud from Kinect, the computationally most expensive processing step is to solve the soft-body dynamics carried out by the bullet physics engine. In particular due to the inaccurate SURF-feature-based model updates, the solving step takes significantly longer in cases of larger paper movements, which explains the large variability of the obtained frame-rate. A significantly faster solving of the soft-body dynamics would only be possible by integrating a GPU-based solver.

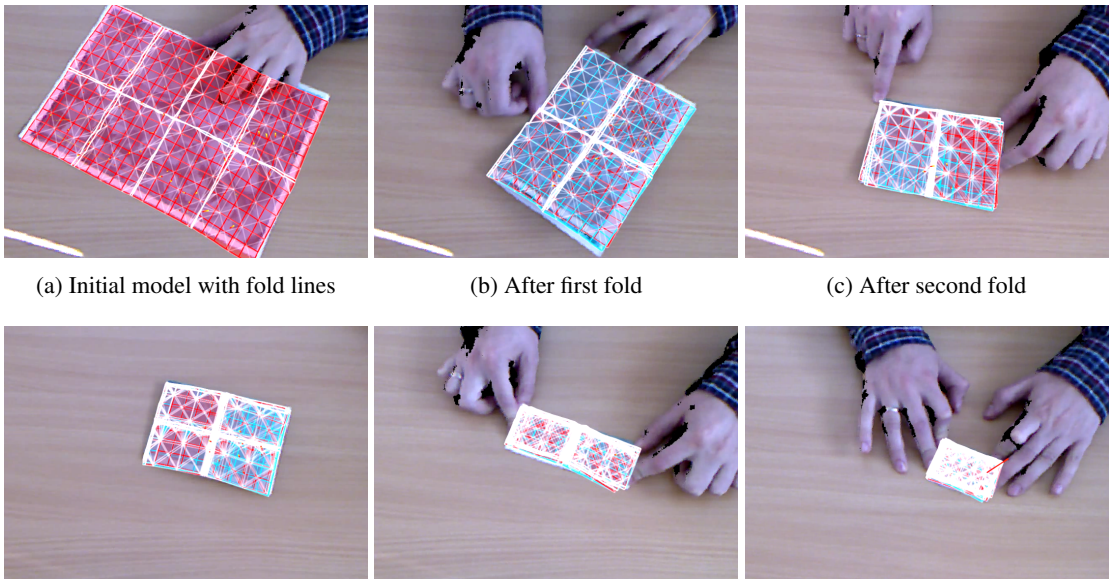
6.3.2 Qualitative Evaluation of Human Folding Sequences

Analogously to the evaluation of the marker-based tracking system (see Section 5.4), the marker-less tracking framework based on a combination of classical ICP and SURF-feature-based tracking was evaluated on the basis of a set of typical human manipulation sequences. The results can be summarized by saying that even though the marker-less tracking system does not completely match the original marker-based approach, its performance gets very close to it. Indeed there are even some less complex folding scenarios in which it even outperforms the marker-based system. Due to stability issues with the underlying bullet physics engine, self-collision handling was disabled for the tests and tracking frame-rate bottlenecks were sometimes manually counterbalanced by deliberately decreasing the manipulation speed.

Iterative Folding

Iterative folding interactions were difficult for the original marker-based tracking system. In particular, a third fold resulting in eight layers of paper stacked on top of each other could not be tracked satisfactorily. Here, the new marker-less tracking system performed significantly better, allowing not only a third fold, but also a forth fold to be robustly tracked (see Figure 6.16).

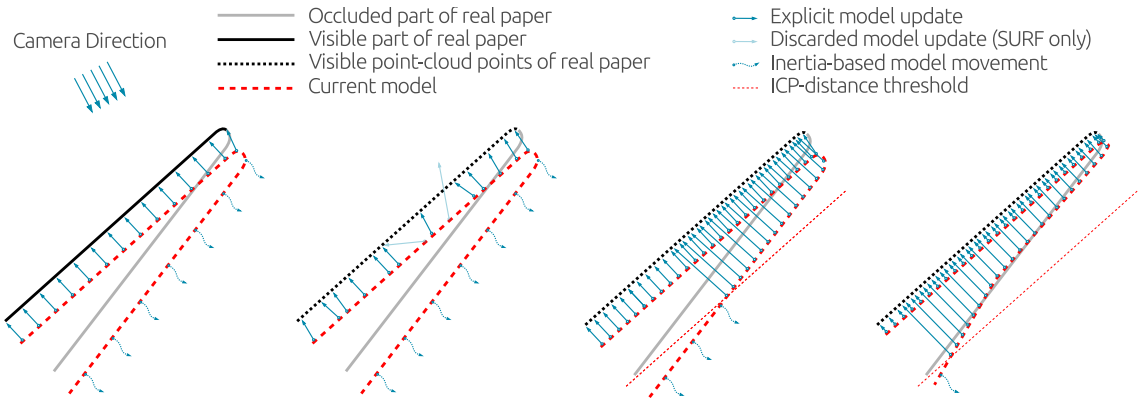
In contrast to the iterative folding experiment with the marker-based tracking system, it was not even necessary to have the system memorize the folds that were applied to the paper. This can partly be explained by the disabling of the physical self-collision of the model that leads to less tension around the folds within the model and by the fact that the generalized model represents the fold lines much more accurately than the grid-based model used in combination with the marker-based tracking system. However, the main reason for the superiority of the point-cloud-based tracking in this case is the fact that the marker-based tracking only provides features for the model update as long as markers are fully visible. Here, in particular the ICP-based model update branch still provides reliable and dense model updates for the tracking. In addition, the ICP-based tracking has an implicit tendency to *glue* layers of the paper together if they are aligned. This effect is explained in Figure 6.17. If the *maker-paper* is folded in half, only the top of the resulting two layers of the paper is visible, resulting in absolutely no updates to the invisible layer. Therefore the movement of the invisible part of the model depends on the physical constraints only, making it necessary to avoid an accidental unfolding of the model by memorizing the fold explicitly. In case of the ICP-based tracking, however, also the theoretically invisible parts of the model parts not seen by the camera are attracted to the point-cloud, so all model layers are moved towards



(a) Initial model with fold lines (b) After first fold (c) After second fold

(d) Extra fold line added (e) After third fold (f) After fourth fold

Figure 6.16: Tracking of an iterative folding sequence. **(a-c)** The first five fold lines are added initially allowing the paper to be folded three times in orthogonal directions. **(c,d)** After the second fold, an extra fold line through all four layers of the folded paper is added, which then allows the paper to be folded two more times. **(e)** The paper is folded in half for the third time. At this stage reliable SURF-features are scarce, so the system mainly relies on ICP-based tracking. **(f)** The paper is folded in half for the forth time while its deformation is still almost perfectly tracked.



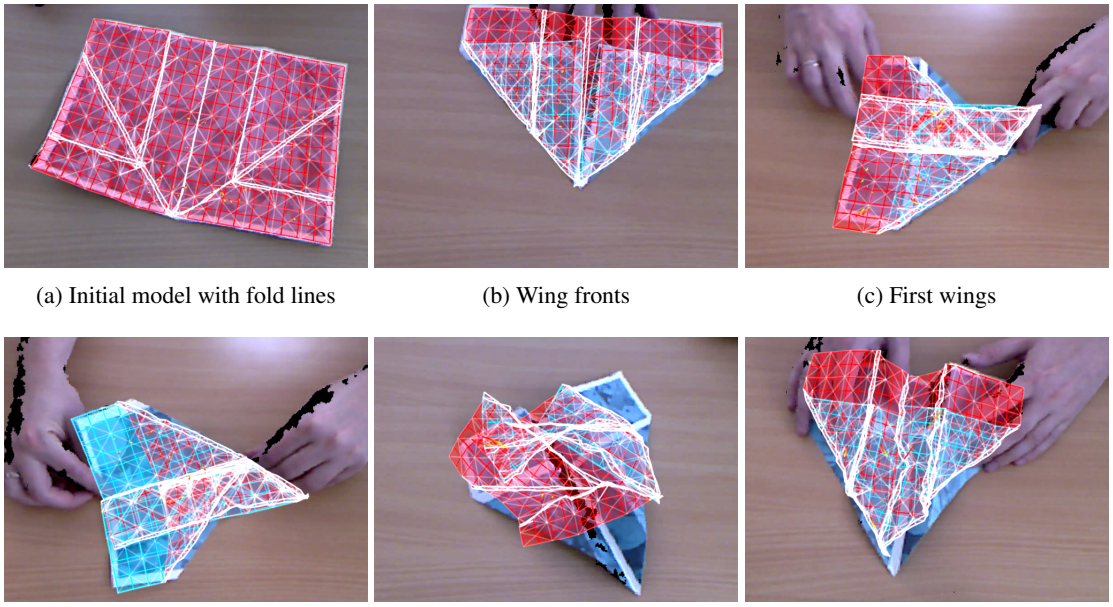
(a) Marker-based (b) SURF-feature-based (c) ICP-based (frame t) (d) ICP-based (frame $t+1$)

Figure 6.17: Schematic comparison of the influence of the marker-based and the point-cloud-based model update mechanisms. In the current situation, the real paper was moved towards the camera. **(a)** Model updates for the marker-based tracking. Here, only model parts that map to actually visible parts of the paper are moved. The remaining model parts are moved according to the physical constraints. **(b)** SURF-feature-based updates. In comparison to (a), the updates are less stable, less homogeneously distributed and some of them are filtered out, but the general update mechanism is similar. **(c,d)** ICP-based model updates. The updates are denser and due to the missing sophisticated link between point-cloud points and model-patches model parts that map to the invisible layer of the paper are attracted towards the point cloud. An internal *ICP distance threshold* that limits the maximum distance for the nearest neighbor search of the ICP allows this effect to be partly suppressed. However, in successive time frames (d), more and more parts of the model are drawn into the threshold region leading to both layers of the model being *stitched together* after a short time.

the same plane. This effect also explains the instability of the physical model when enabling self-collision, which would permanently try to counteract the model updates in such configurations.
This gluing effect leads to severe issues for the marker-less tracking approach when the paper is unfolded. Here, the two model update branches (ICP/SURF) partly work against each other during unfolding. While the SURF-feature-based branch yields correct updates to actually separate the layers of the paper, the ICP-based updates hold the paper layers together. When undoing the last of the performed iterative folds, only a few SURF-based updates are available and these are furthermore relatively small as the actual displacement of surface coordinates when folding in half is halved with each iterative fold. At the same time, however, the ICP-based updates are not affected, so they keep their strength. A possible heuristical solution to this drawback would be an automatic adaption of the controllers' proportional gains for the two update branches, but this is left for future work.

Folding a Paper Aeroplane

The aeroplane folding experiment (see Figure 6.18), adds an additional level of complexity to the detection and modeling framework. While the necessity to model diagonal folds was only more difficult for the earlier regular grid based model, the more complex interaction itself yields the major difficulty increase for the new tracking system. The aeroplane folding sequence contains several transient states and the final shape contains many types of folds. The folds that abduct the wings from the fuselage are approximately 90 degrees and the bottom fold of the fuselage is approximately 130 degrees and need to be modeled. In addition, the folding sequence contains a 180 degree out-of-plane rotation of the deformed paper.
Before starting to fold the paper, all fold lines that are needed for the final paper aeroplane are explicitly added (see Figure 6.18a). The first manipulation steps (see Figure 6.18b) pose no major challenge for the tracking system. The diagonal fold lines of the wings' fronts are manually memorized and added to the model. When folding the wings (see Figure 6.18c), the number of good SURF-matches found is sufficient to separate the two model wings correctly. Before the second wing can be folded (see Figure 6.18d), the whole partly folded aeroplane must be tracked while it is turned upside down, which is handled very well by the system. However, after folding the second wing, the tracking of the final construction steps of the aeroplane by rotating and adapting the wings' fold angles fails completely (see Figure 6.18e). This step is particularly difficult for the system for a combination of reasons. First, the large out-of-plane rotation in parallel to abducting the wings is a difficult task, which gets even harder by the large amount of occlusions caused by the two manipulating hands. In addition to this, the paper is already in a complexly folded configuration state exposing only smaller surface patches to the camera, which not only hardens the SURF-feature-matching significantly, but, due to new combined surface patches resulting from the folding, also increases the likelihood for pseudo-features occurring randomly during the interaction. Moreover, for the combined ICP/SURF-based tracking, the abduction of the wings is particularly difficult because of the ICP-based tracking's tendency to keep paper layers (here the fuselage and the wings) together once they were co-planar.
Only by manually spreading the finished aeroplane (see Figure 6.18f) in order to expose a major fraction of its upper surface, the system is able to recover and track the paper's deformation correctly. In this configuration, a significant portion of the difficulties that arose from the putting together the aeroplane were nullified. However, it must be mentioned that even though the tracking recovered to an acceptable level of inaccuracy, subsequent tracking was very unstable and likely to fail again when the paper was minimally adapted. Here, the original marker-based tracking system yielded much more robust results.



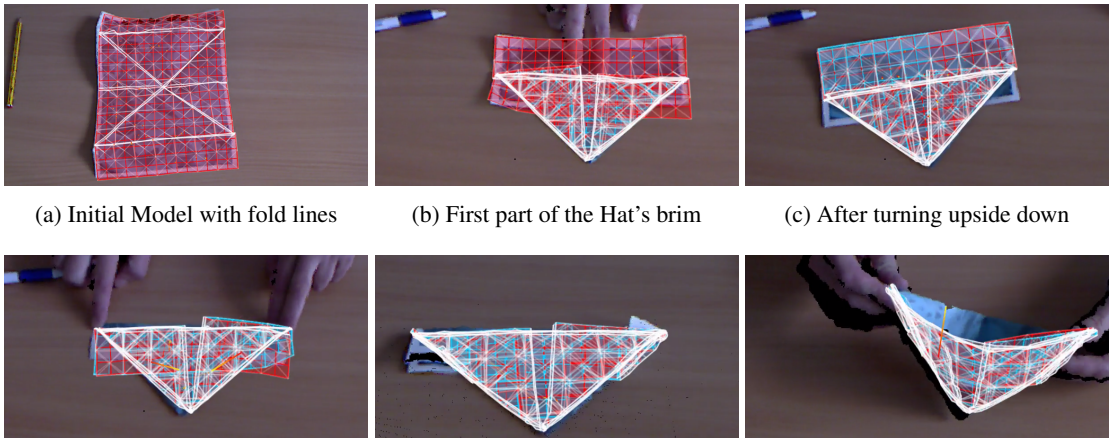
(a) Initial model with fold lines (b) Wing fronts (c) First wings

(d) Second wing (e) Severe issues (f) Improved with spreading

Figure 6.18: Tracking an aeroplane folding sequence. **(a)** Initial paper model with explicitly added fold lines. Each fold line was added twice to increase the model flexibility around the folds. **(b)** The front parts of the wings were folded and the folds are memorized. **(c)** After folding the paper in half the first wing is folded back. **(d)** Before the second wing can be folded the paper is turned upside down. **(e)** After finishing the last fold, the final paper aeroplane is turned the right way up. Here, severe tracking issues occur and the system is not able to automatically recover. **(f)** By spreading apart the wings, the deformation becomes less complex and more SURF-features become visible, which allows the system to recover its tracking of the object. However, it must be admitted that this final configuration is very unstable even if the object remains motionless.

Folding a Paper Hat

The paper hat folding experiment adds some further aspects to the aeroplane folding sequence. It allows us to focus on more subtle aspects emerging from small paper faces being folded and furthermore it underlines an additional class of limitations of the tracking system. Similar to the paper aeroplane folding experiment, the first folding steps in which the paper is only moderately folded does not pose a large challenge for the system. Starting with an unfolded model with prepared fold lines (see Figure 6.19a), the initial folding in half, the two diagonal folds of the hat's top part as well as the first part of the hat's brim are tracked robustly without any significant issues (see Figure 6.19b). However, while turning the partly folded model upside down in order to prepare it for the folding of the second part of the brim, the already folded part of the model's brim becomes erroneously unfolded (see Figure 6.19c). This is due to the missing memorization of the brim's fold line, which is caused by the technical difficulty in our mouse-based approach to select single folds by hovering them in the 2D image space. Actually the deactivated model self-collision allows the system to perfectly recover from this by letting both layers of the paper brim follow the folded one (see Figure 6.19d). A more severe issue of the tracking system occurs in the next folding step. While in contrast to the marker-based tracking system, the ICP-based tracking branch allows the brim-corner folds to be tracked (see Figure 6.19e, right), it is also prone to accidentally *believe* that a small patch of the paper was folded, even though it was not (see Figure 6.19e, left). The main reason for this is the small inertia and the lack of reliably matchable SURF-features on such small model patches, which is even compounded by the fact that such tracking



(a) Initial Model with fold lines (b) First part of the Hat's brim (c) After turning upside down

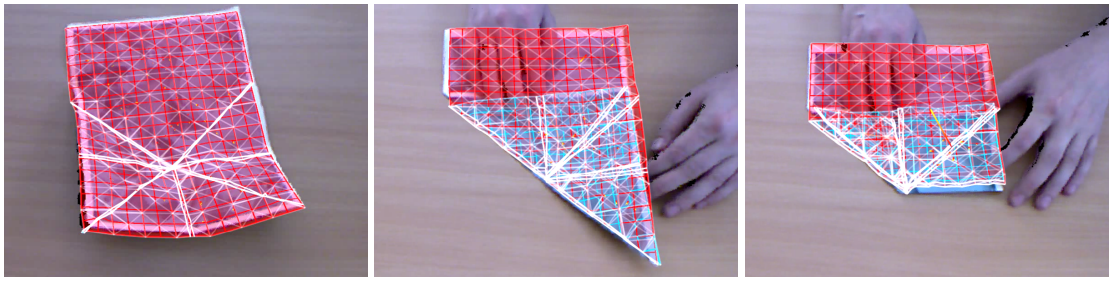
(d) Second part of the brim (e) One corner bent around (f) Final model tracking

Figure 6.19: Tracking of a human folding a paper hat. **(a)** Model endowed with all fold lines needed, except for the final small corners of the brim. **(b)** All initial steps, such as folding in half and performing the two diagonal folds were robustly tracked. The tracking of the first part of the hat's brim was also successful but as the resulting fold is co-linear with the still to-be-folded part's fold line, the fold cannot be memorized. **(c)** After turning the paper upside down, tracking of the already folded part of the brim was lost. **(d)** The folding of the second part of the brim also brings the lost first part correctly into position. After this, the fold lines for the brim's corners are explicitly added. **(e)** While folding the corner on the right hand side of the image, the other model corner appears erroneously folded as well. **(f)** The final configuration can be tracked, but the resulting model behaves like a flat surface. Its underlying cone shape is not modeled correctly.

errors cannot be undone in hindsight as the ICP-tracking continuously moves the small patch along with the actual layer of the paper it was folded onto.
Even though this error coincidentally reflects the desired target paper configuration in the present example, its underlying explanation must be emphasized as one of the most severe drawbacks of the proposed tracking approach because it is also responsible for the fact that the two faces of the hat's top cannot be separated to form the hat's actual cone shape. Figure 6.19f shows that the layers of the paper model behave like they are glued together.

Performing a Squash Fold

The tracking of an origami squash fold is more difficult for the system with regard to the physical modeling than to the actual visual detection. As model self-collision had to be explicitly deactivated due to stability and performance issues within the bullet physics engine, the physical modeling is much more likely to fail here. After preparing the sheet of paper and the model by adding fold lines to the model (see Figure 6.20a) and performing the first diagonal fold (see Figure 6.20b), the actual squash fold is performed. However, even though the deformation of the paper is initially tracked satisfactorily, the final model configuration is wrong (see Figure 6.20c) as the actually to be squashed part of the paper penetrates to the top layer. Thus, the model part ends up on the, from the viewers perspective, left hand side of the model rather than between the two paper layers on the right hand side. A fix that makes such interactions robustly trackable would necessitate solving the stability issues with the physics engine. Since the stability issues are a direct result of the implemented ICP-based tracking branch, it can be concluded that the squash fold discipline was accomplished better by the marker-based tracking system.

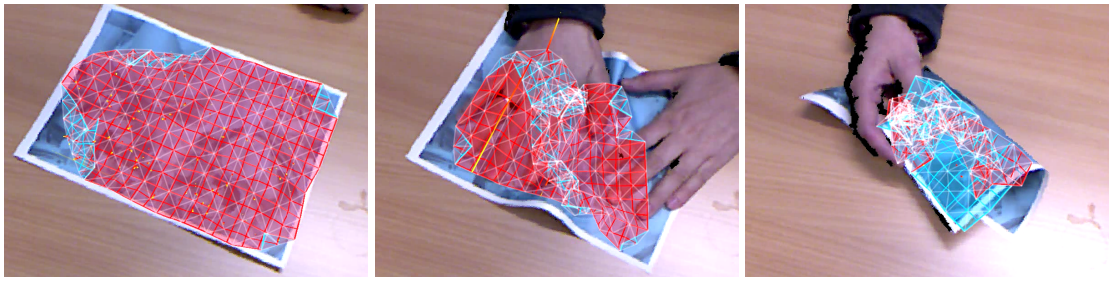


(a) Initial model (b) Before squash fold (c) After squash fold

Figure 6.20: Tracking a human performing a squash fold. **(a)** Initial model prepared with all fold lines needed. **(b)** First diagonal fold applied. **(c)** Final paper model after performing the squash fold. Even though the general shape of the real paper seems to be approximated well, the tracking of the folding operation was not successful. The doubled appearance of a diagonal fold on the right hand side of the image shows that the inner pocket erroneously penetrates to the top layer while the squash-fold is performed.

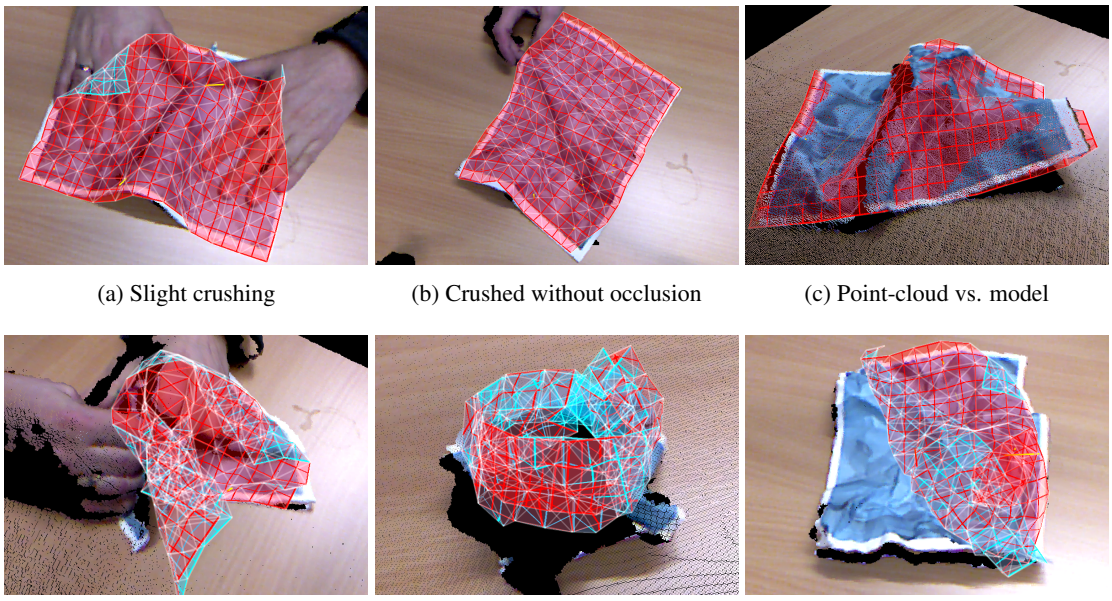
Crushing Paper

As the final experiment, also a paper crushing interaction was monitored, which has a stronger randomized element and therefore requires significantly more from the tracking and modeling system. While in the previous folding experiments the target model state was defined in a precise fashion, which was accomplished by initially adding paper fold lines, the crushing of the paper is supposed to deform all paper patches with equal likelihood. Therefore, the model cannot simply be prepared with a given set of fold lines. Instead, in order to allow all parts of the paper to be locally bent with a high curvature, the global model stiffness must be decreased. Due to the fact that the stiffness coefficients $s_{ij} \in [0, 1]$ (see Section 6.1.1) of the model's bending constraints internally used by the bullet physics engine do not have a well defined or documented unit, which would us allow to infer reasonable values for the global model stiffness, the value was manually tuned in a trial and error manner. An initial attempt to allow for an almost arbitrary curvature to be modeled by using an extremely low global stiffness of $s_{\text{low}} = 0.01$ emphasized the importance of the internal model stiffness for the whole tracking framework. Figure 6.21 shows that the low stiffness value leads to a permanent underfitting and agglutination of the model faces, which in turn yields a bad tracking performance. Further tests showed that a global stiffness coefficient of $s_{\text{medium}} = 0.2$ provides a good trade-off between large local curvatures and preventing the model from agglutinating. Figure 6.22 shows the outcome of the crushing experiment conducted using s_{medium} as the global stiffness value. While lighter crushing manipulations (see Figure 6.22a-c) are tracked very well and also quantitatively correct, the result of stronger crushing (see Figure 6.22d-e) is usually only replicated qualitatively well by the tracking system. However, in situations such as (see Figure 6.22e), it has to be admitted that humans would also naturally switch their internal model from a distinct deformation described by the actual shape of the paper surface, to a more general representation, most likely referenced by a token, such as *a crushed paper sphere*.
Once the paper is heavily crushed, the paper deformation cannot simply be reverted. Instead, an unfolding only flattens out the paper coarsely but the surface remains wrinkled (see Figure 6.22f). This leads to an extra difficulty for the SURF-feature based tracking component. Even though the qualitative change of the paper surface texture is not very obvious for the human eye, the spatially varying lighting conditions of the wrinkled paper patches lead to a strong change of the SURF-feature descriptors that are computed. This leads to a severe decrease of usable SURF-feature matches, which, in turn, weakens the SURF-feature-based tracking branch significantly in such situations.



(a) Undeformed paper (b) Slightly crushed and occluded (c) Crushed and bent

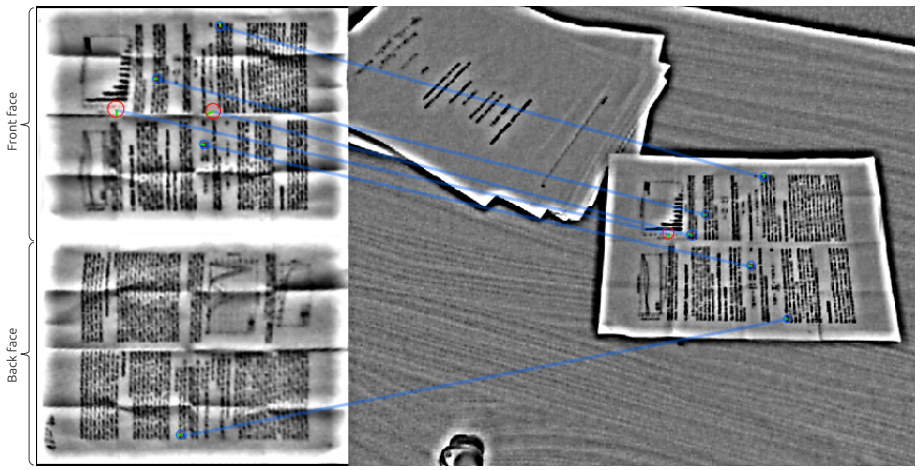
Figure 6.21: Paper crushing manipulation carried out using a *too* low global model stiffness of $s_{\text{low}} = 0.01$. **(a)** Even in case of the undeformed paper, the model corners have a tendency to curl inwards. **(b)** Parts of the model that once have been aligned are unlikely to become detached at a later point in time, leading to a shrinkage of the model. **(c)** In the course of the interaction sequence, more and more parts of the model are agglutinated into a small pack.



(a) Slight crushing (b) Crushed without occlusion (c) Point-cloud vs. model

(d) Stronger crushing (e) Crushed paper sphere (f) After unfolding

Figure 6.22: Paper crushing manipulation carried out using the manually optimized global model stiffness of $s_{\text{medium}} = 0.2$. **(a)** Initial slight crushing while the paper is severely occluded by the manipulating hands. The tracking of the surface deformation works sufficiently well, except for the occluded parts. **(b)** Once the occluding hands are removed, the deformation of the whole sheet of paper is well reflected by the model. **(c)** For a more detailed comparison, the situation shown in (b) is visualized again, but from another perspective. Here, the model is not rendered on top of the scene, but, in order to visualize what is on top of what, into it. The differences between the model and the point cloud are not critical, but some smaller structures such as the actual shape of the center bending of the paper are smoothed out by the model. **(d)** Stronger crushing of the paper in order to make a paper sphere. The visible parts are still tracked. **(e)** Final paper sphere. Due to the extreme occlusions, the deformation can only be judged in a qualitative fashion. **(f)** If the paper was heavily crushed once, the variable lighting of different paper surface patches distorts the actually seen surface texture too much. Therefore, most SURF-features cannot be matched anymore, making the unfolding of the model much more difficult for the system.

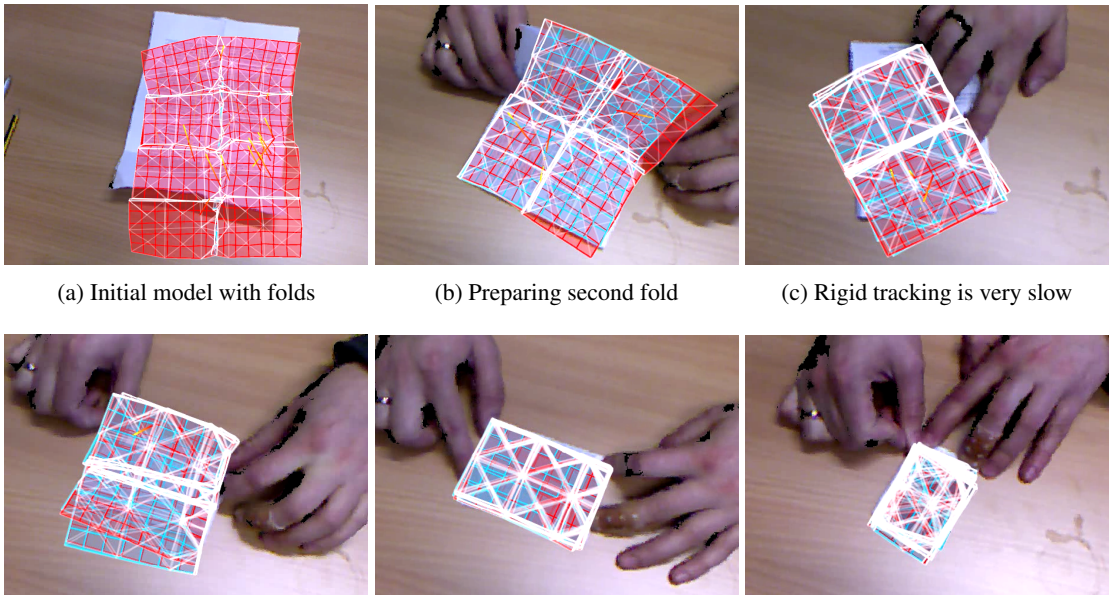


(a) Paper texture maps (b) Detected paper

Figure 6.23: Visualization of SURF-feature matches in case of using realistic paper textures for the front and back face. The fact that the fold lines of the paper and the prototypes are amplified by the local threshold enhancement operation does not effect the results significantly.

6.3.3 Tracking Folding of Common Textured Paper

The marker-less tracking system that combines ICP with SURF-featured based tracking was shown to provide very good tracking results. However, the important initial assumption of a segmentation and SURF-feature *friendly* paper texture cannot easily be extended to real-world scenarios. Thus, another important question is how well the tracking system can deal with a more realistically textured sheet of paper. A blank sheet of paper would naturally provide no SURF-features for the tracking and thus the results would be identical to the results presented for the tracking approach based only on ICP (see Section 6.2). As pure printed text will, due to the limited texture resolution visible in the VGA Kinect color camera images, also provide very unreliable SURF-feature matching results, a double sided research paper page was used. The two pages (Page 3 and 4 from [Rublee et al., 2011]) used for the front and the back-side texture, contain text, formulas, enumerations and a few diagrams. By using the contrast enhancement operator presented in Section 6.3.1 the coarse structure of the paper, amplified by the two-column layout and the different paragraphs, yields a small, but representative, number of reliable SURF-feature matches (see Figure 6.23). The figure shows five correct and one incorrect matches, i.e. this represents, in comparison to the previously introduced *SURF-feature-friendly* texture, an order of magnitude less features (see Figure 6.13). However, many matches are very fragile, i.e. they are likely to occur and to disappear in subsequent processing frames, leading to an at least two times lower number of actually used SURF-feature matches over time. Even though the average error percentage of about 20% of the matches did not change in comparison to the optimized texture, the fraction of actually used matches needs to be decreased to about 50%. This is due to the fact that in both cases an occasional occurrence of three to five erroneous matches seems to have remained equally likely, which is, however, a much higher percentage of all matches in the present case. In addition, the smaller numbers of features must be partly compensated by increasing the proportional gains for the SURF-feature-based updates, which, in turn, makes the tracking more prone to become compromised by feature matching errors.
In the present scenario, the increase in segmentation difficulty does not carry weight as a simple adaption of the segmentation lookup table (see Figure 6.6c) is sufficient to separate the white and bright gray paper pixels from the background. When using the system in a more realistic scene, where other objects are also white, a more sophisticated segmentation method, which not only relies on color information would be needed. An automatic and model-free point-cloud-based scene

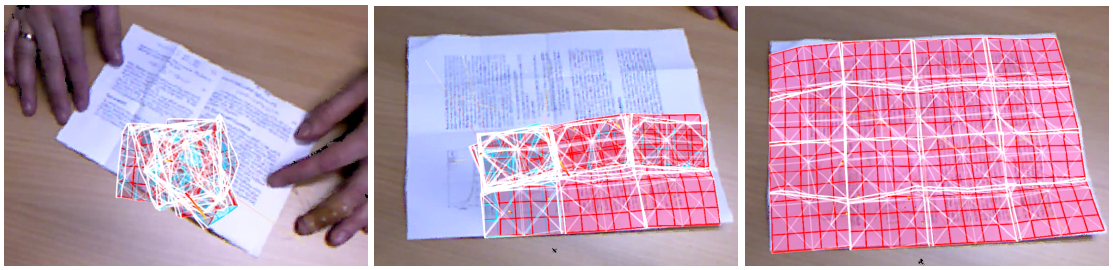


(a) Initial model with folds (b) Preparing second fold (c) Rigid tracking is very slow

(d) Error while tracking 3rd fold (e) Repeated 3rd fold successfully (f) 4th fold added and performed

Figure 6.24: Folding a realistically textured sheet of paper. **(a)** Initial paper model. The fold lines of the first three folds are already added. The red-to-orange gradient lines indicate that only a few SURF-feature-matches are used. A single but steady SURF-feature matching error yields unstable tracking results. **(b)** After successfully tracking the first fold, the second fold is already in preparation. **(c)** Due to the small number of SURF-based updates, the in-plane tracking of paper operates very slowly. **(d)** When applying the third fold for the first time, the manipulation was carried out too quickly, so the tracking was not able to follow the deformation of the paper fast enough. **(e)** In the second trial, the third folding movement was carried out more slowly, so that it could be successfully tracked. **(f)** After adding an additional fold line though all layers of the folded model, the fourth iterative fold could be tracked accurately.

segmentation, such as [Ückermann et al., 2013] could be used here.
Figure 6.24 shows the course of an iterative folding sequence, carried out using the realistically textured paper. The first three fold lines are initially added to the model. As it can be seen from the SURF-feature matching indicators (red-to-yellow gradient lines in Figure 6.24a), only in the order of 10 SURF-matches are actually used, which is why larger movements of the whole paper can only be tracked slowly. With each iterative fold applied to the paper, the contribution of the SURF-feature-based model update mechanism is decreased as more and more SURF-features disappear. However, as this was also the case for the former SURF-feature-friendly paper, the results of later folding stages are very similar. Thus, the tracking system is also able to robustly track the second and the third iterative fold directly. After manually adding the fourth fold line at run-time (after Figure 6.24e), the fourth fold is also successfully tracked.
However, due to the lower density of SURF-feature matches, the tracking of an unfolding of the paper becomes more problematic. Figure 6.25 shows the results of an exemplary experiment, that started at the end-state of the previous folding experiment (see Figure 6.24). Even after reverting the first three folds (see Figure 6.25a), the system is not able to unfold the model appropriately. Only when finally undoing the last fold (see Figure 6.25b) the model starts to partially unfold as well. However, once again the model does not even get close to the real unfolded paper. This effect is explained by the ratio of the proportional gains of the two model update branches SURF and ICP. The gluing force from the ICP-based updates outperforms the accumulative forces from the SURF-feature-based updates, so the model does not unfold any further. By manually altering the proportional gain ratio in favor of the SURF-feature-based update branch, the model unfolds



(a) Model remains agglutinated (b) Modeling still very inaccurate (c) Enhanced SURF-based updates

Figure 6.25: Unfolding realistically textured sheet of paper. **(a)** Even after reverting all but one fold from the originally four times folded sheet of paper, the gluing characteristic of the ICP-based model update branch still outperforms the SURF-based updates that *try* to unfold the model. **(b)** Even when unfolding the paper completely, the tracking is very inaccurate. In this situation, both the small number of found SURF-matches, as well as the decreased ratio of used SURF-matches, do not suffice for more accurate tracking. **(c)** Only after manually adapting the ratio of the proportional gains used for SURF-based updates and ICP-based updates respectively, is the tracking restored.

correctly (see Figure 6.25c). However, this manual adaption is not suited for permanent use as the amplification of the SURF-feature-based updates creates a heavy jitter in model leading to bad tracking results and lower tracking frame-rates.
As a more permanent heuristical solution, it would be necessary to integrate an automatism that only temporarily amplifies the SURF-feature-based updates if the discrepancy between the model and the observation is high. This, of course, leaves the question of how to estimate the discrepancy and it also adds a larger set of additional parameters, such as a discrepancy threshold, the length of the time window for the amplification and the actual strength of the amplification.

6.3.4 Discussion

The presented combination of ICP and SURF-feature based tracking yields promising results even when not using explicit *SURF-feature-friendly* textures. An intrinsically emerging and welcome property is the seamless automatic interpolation between using SURF-feature updates and working in ICP-only mode. SURF-features that are detected enhance the system significantly but even in the complete absence of reliable SURF-features, the system continues to work. Regarding the introduced *non SURF-friendly* textures the use of common printed sheets of paper that show mostly text and whose global appearance structure is mainly distinguished by vertical paragraph structures only randomly provides useful SURF-features and therefore the system switches to mostly ICP-only mode.
In addition to increasing the camera resolution to detect a larger amount of reliable SURF-features, points in the point-cloud that correspond to paper-edges could be used. However, initial tests showed that the identification of edge-features is a very complex task, in particular in the presence of heavy occlusions.
An obvious question is whether our tracking system scales to other deformable objects or indeed can track rigid objects. Even though this would require a fundamental re-organization of the implemented tracking framework, it is very likely that the system would perform well in both cases. Extending our approach to other deformable objects in theory only requires the model to be adapted. While the tracking of comparably stiff deformable objects, such as a pillow would intrinsically be very similar to tracking paper, softer deformable objects, such as a piece of laundry would necessitate larger adaptions to the model.
In contrast to this, the tracking of rigid objects is likely to be managed even better by the system. In

particular box-shaped objects, or objects whose shape can be approximated well by a combination of such shapes, could be represented by a much smaller number of nodes and faces. This would lead to a significant reduction of the number of constraints that are needed to increase the model stiffness, which would, in turn, allow for a much more efficient tracking run-time. In contrast to common ICP-based or feature-based tracking systems that use non-physical rigid object models, the use of the physics engine would still provide a set of most welcome implicit features, such as the simulation of collisions and automatic temporal tracking constrained with physical-plausibility.

6.4 Automatic Fold Detection and Optimization

The necessity to manually add folds accurately is a severe limitation of our tracking and modeling system. Even though the proposed mouse-based interaction makes it very easy to add fold lines in a drag-and-drop fashion, an automatic detection mechanism that could optionally be primed with the definition of a coarse target fold line, would enhance the system significantly. The problem of automatic fold detection can be split into two parts:

1. Fold onset detection
2. Fold geometry estimation/optimization

In order to solve the first part, a mechanism is needed that is able to distinguish whether a given paper configuration (optionally supplemented with the temporal configuration history) shows a fold has been added to the model. However, exhaustive tests revealed that the creation of a robust estimation system based on the paper configuration alone is a very difficult task to achieve (see Section 6.4.2). In particular the discrimination between *heavy bending* and *actually folding* seems – even for a human – very difficult without taking into account further sources of information, such as tracking the manipulating hand motions. If a robot is used, information about *when* a fold is being attempted can simply be assumed to be given. In order to relax the necessity to manually add fold lines to the model using mouse-based input when monitoring human folding sequences, a simple speech-based triggering mechanism that allows the human manipulator to keep his hands on the paper permanently could be added trivially. From the perspective of a human expert explaining how to fold a certain origami figure, it would also be quite natural to supplement the observation of the folding procedure with hints such as *"ok, now we fold here"*, *"now I have to undo that fold"* or *"next, this part has to be bent, but we have to take care not to fold it"*.
In order to solve the second part, the system must be enabled to estimate the geometry of a fold line. Assuming that the onset for the adding of the fold line is given, the system has to provide the paper-space coordinates of the to-be-added fold line. Optionally, it should be possible to provide an *educated guess* of the new fold line to narrow the search space for the system. This educated guess could either be given by the robotic manipulation sequence, in which the coarse geometry of the to-be-achieved fold line is known or it can be calculated on the basis of the perception of either the model deformation alone or by employing other features. It is important to account for the fact that the onset detection might be subject to errors so the geometry optimization must also feature a rejection mechanism.
In this Section, first the coarse structure of a particle-based prototype system is presented (see Section 6.4.1). In Section 6.4.2 experiments that were conducted towards automatic fold onset detection are presented. Even though no satisfactory solution could be found, the section provides important insights into the inherent difficulties of this problem and in particular it explains why Gaussian curvature computed on the model does not suffice. Subsequently, a system for the automatic estimation of a fold's geometry is presented (see Section 6.4.3) and is qualitatively evaluated (see Section 6.4.4).

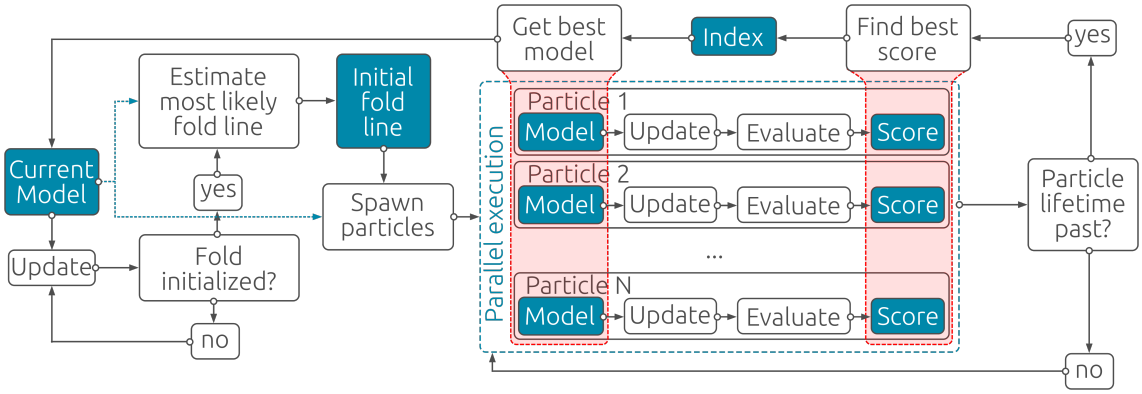


Figure 6.26: Control flow diagram for the automatic fold detection prototype system. Starting with an initial current model (left), the system remains in the *normal* operation state until the *Fold initialized?* component detects that a fold line is potentially being added to the paper. In this case, based on the current model-surface deformation, a *most likely* fold line is estimated and then used to enter the second operation state that is based on a particle set. After a given number of processing frames, the system automatically selects the best performing particle in order to go back to the original single model operation mode.

6.4.1 A Prototype System

The developed prototype system for automatic fold detection employs two major operating states (see Figure 6.26). In the first state, the system operates like the previously presented model tracking system, where the model is updated and physically simulated using the perceptual belief in each time frame. On top of this, the fold onset detection module is inserted. If the adding of a new fold is detected/triggered, the system enters the second operating state. Here, a particle system is instantiated, in which each particle represents a hypothesis about the actual geometry of the potential fold line. The particles are randomly distributed around a heuristically estimated first guess that is based on the evaluation of the paper surface deformation at the instantiation time of the particle system. In order to allow the system to reject the newly added fold line, one of the spawned particles represents the paper without a new fold line. The no-fold particle's weighting is explicitly increased to account for the fact that a more flexible model with an erroneously added fold line would not automatically provide worse modeling results if the real paper was not actually folded. From a more general perspective, this could be seen as a penalization of high numbers of folds. Once the particles are spawned, the resulting set of models – one for each particle – has to be tracked in parallel, each in a dedicated physics world to avoid unwanted inter-model collisions. In the implementation, in order to avoid a resulting processing frame-rate drop directly proportional to the number of particles, each particle is explicitly handled in a dedicated thread. The rating scores of the particles are computed and integrated over time, yielding a historical particle score that is used to dissolve the particle system after a given number of time-steps. Here, the model that corresponds to the best performing particle over time, i.e, the one with the highest temporally integrated score, is used to replace the original *current model*. Further common variants of particle systems, such as *Condensation*-based [Isard and Blake, 1998] hierarchical iterations or joining the resulting particles with respect to their score could be implemented in future work.

6.4.2 Fold Onset Detection

In order to automatically detect that the observed paper has been folded, a local paper surface curvature estimator seems like an obvious choice. However, as the best guess available about the actual paper surface deformation is the shape of the current model, we face a so-called *chicken and*

egg problem. At the time, when the real paper is folded, the current model is not endowed with a corresponding fold line. Therefore the model will, due to the underlying physical model's stiffness constraints, not reflect the fold, but simply have a smooth bend, which in turn makes it impossible to detect the fold on the basis of the model deformation alone. In order to explain this issue in a more detailed fashion, a common surface curvature operator and the *paper surface curvature map* that results from applying that operator to each model node is presented.

Gaussian Curvature for Discrete Triangle Patches

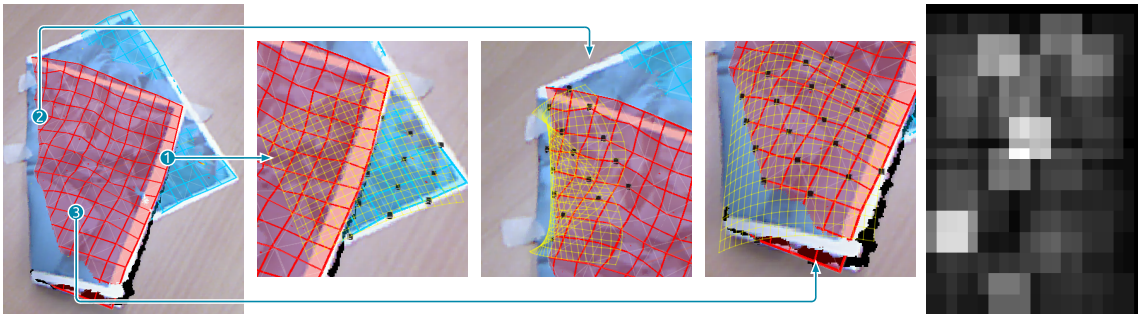
A very common surface curvature quantity is Gauss curvature [Sullivan, 2005]. While for smooth surfaces there exists a precise definition, several equally accepted definitions are available for discrete surfaces that are represented by triangle patches. A very simple method to approximate the Gauss curvature κ_g at a vertex v_i of a surface defined by a triangular grid is defined by $\kappa_g = (2\pi - \sum_j \alpha_j)/(\frac{1}{3}\sum_j A_j)$ [Calladine, 1986; Gray et al., 1998]. Here, the sum index j runs over all triangles spanned by the direct neighbor nodes of v_i, α_j is the angle subtended at v_i by each triangle and A_j is the corresponding triangle area [Razdan and Bae, 2005]. The result is non-negative, and only 0 if v_i and all neighbors j are co-planar. However, an initial test implementation confirmed the statement by Razdan and Bae [2005] that the approximation is *"not necessarily accurate under all circumstances"*. In particular a non-uniform surface triangulation as well as the finite edges of the paper surface lead to a non-equal number of neighboring faces per vertex, which in turn has a very strong influence on the resulting curvature values. Therefore, commonly more complex continuous methods are used. These approximate the discrete surface locally by a parametric continuous surface model, whose Gaussian curvature can be computed analytically.
For the introduction of the *paper model surface curvature-map*, a method based on bi-quadratic Bézier surfaces, $x(u,v) = \sum_{i,j=0}^{2} b_{i,j} B_i^2(u), B_j^2(v)$, was implemented. The method was proposed by Razdan and Bae [2005] and it was shown to outperform previous methods. While the actual fitting of the Bézier control point parameters $b_{i,j}$ is performed in a standard least-squares manner extended with a surface smoothing parameter, the preparation of the input data was of a more complex nature that came up with a non-negligible implementational expense.

Fold Detection Based on a Gaussian Curvature Map

The creation of the Gaussian curvature map was implemented by approximating the surface around each paper model node with a bi-quadratic Bézier surface. For the fitting, a node's double-star[9] neighborhood was used (see Figure 6.27b-d) as suggested by Razdan and Bae [2005]. As it turned out that using the Gauss curvature at the center of the resulting Bézier patch provided almost random results, a patch's maximum Gauss curvature[10] was used instead. However, as Figure 6.27e shows, even a morphological dilation and an additional exponential temporal smoothing did not yield very convincing results. A large set of other heuristical adaptions, such as using a node's star or the triple-star neighborhood, different 2D-filters, such as median, Gaussian smoothing and different kinds of morphological operations was also implemented and tested, but did not improve the results.
Further investigations lead to the insight that this failure is caused by two main reasons. The first reason is the above mentioned *chicken and egg* problem that the method can only detect a fold that

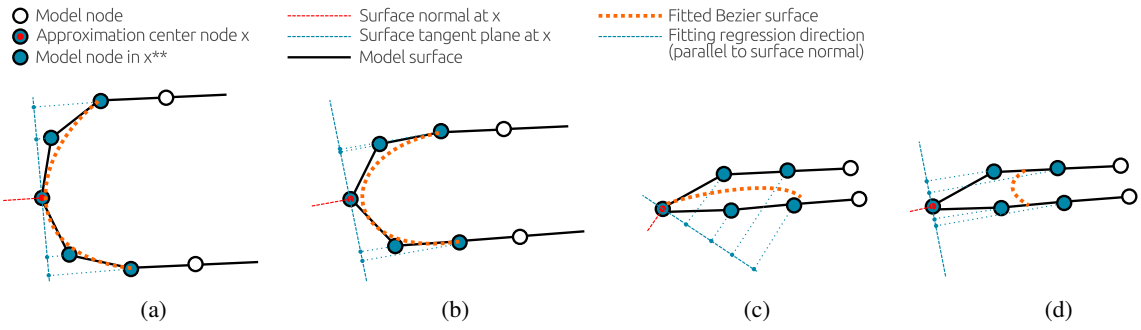
[9] Representing the triangle grid as a set of nodes $\{x_i\}$ and a set of triangles $\{(a,b,c)_j\}$, the star of x_i (or simply $x_i^\star$) is defined by $\{x_k | \exists (a,b,c)_j : i,k \in \{a,b,c\}\}$, i.e. the set of nodes directly connected to x_i by one of the triangle edges. The double star $x_i^{\star\star}$ is then defined as $\cup_{x_j \in x_i^\star} x_j^\star$ and the triple star is defined analogously.

[10] Estimated by evaluating the patch's Gaussian curvature at 10×10 regularly sampled positions



(a) (b) (c) (d) (e)

Figure 6.27: Bi-quadratic Bézier approximation of the paper model surface and resulting Gaussian curvature map. **(a)** Point-cloud image with augmented paper model. While the real paper is actually folded, the model's bending stiffness does not allow a sharp fold to be reproduced, so the model appears bent only. **(b-c)** Examples for the bi-quadratic Bézier approximation centered at selected model vertices. For the Bézier fitting, the double-star neighborhood (black vertices) of the selected nodes is used. In contrast to the well fitting regression of flat (b) and curved (d) model regions, the region next to the fold (c) reflects only the curvature of the smoothly curved model, but not the curvature of the actually folded paper. **(e)** Dilated and temporarily averaged Gaussian curvature map.



(a) (b) (c) (d)

Figure 6.28: 2D-visualization of the Bézier surface approximation used for the creation of the Gauss curvature map. For a model node (marked red), the local surface is approximated by a bi-quadratic Bézier curve (dotted orange line). In cases, where the actual model curvature is comparably low (a,b), the surface normal (red dashed line) is very accurate and also the estimated curvature is low. In contrast to this, in cases with a high local curvature (c,d) the surface normal calculation is less accurate leading to more or less random Bézier approximations with also random Gaussian curvature estimates.

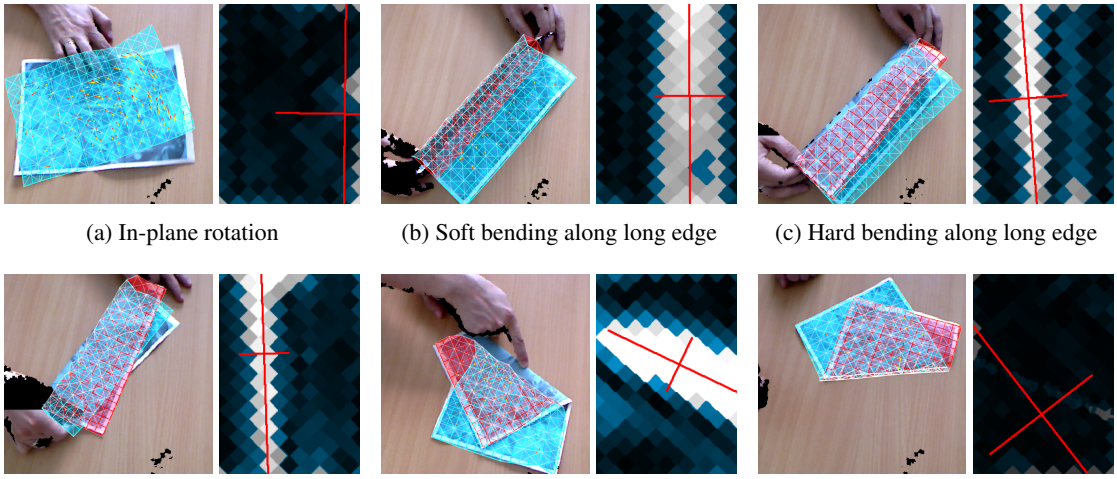
has already been added to the model. Even though this issue is severe, it is not completely sure whether the system would perhaps still provide acceptable results if the model's global stiffness was lowered appropriately. However in that case, the second issue is responsible for bad curvature approximation results. This issue is originated in the curvature computation that internally needs to transform the 3D paper model vertex data into training data for the Bézier surface fitting. As the Bézier surface is a scalar function defined on a 2D input space, the model points must be locally projected onto a tangential plane to the model surface (see Figure 6.28). The plane normal is provided by the physics engine, but results showed that higher actual local curvatures yield higher noise values in the normal estimation, which then in turn result in randomized curvature map entries (see Figure 6.27e). This effect could only be overcome by using a much higher node density for the physical model, which is, however, due to the associated increased of the computational complexity not possible.
The results indicate that the used local model surface curvature estimation method is not the right approach to automatic fold detection. There are, of course, plenty of possible next directions, such as replacing the local Bézier surface approximation or even the whole Gaussian curvature estimation method, employing other directly available properties from the physical model, such as the node velocities or even trying to extract additional information from the input point cloud data directly. As also further endeavors that were made towards finding an appropriate replacement for the Gaussian curvature map did not yield promising results or clues, the problem remains unsolved. However, the described course of the implementations and experiments yields valuable insights into the underlying complexity of the problem.
The fold geometry optimization method presented in Section 6.4.3 uses a manual trigger for adding folds.

6.4.3 Fold Geometry Estimation

For the proposed particle-based prototype system (see Section 6.4.1), the fold geometry estimation module is needed to automatically compute an initial guess of the actual fold, which is then used as the center for the particle distribution. Assuming that no prior information about the geometry of the to-be-added fold is available, the initial guess has to be derived from the properties of the current paper model. The insights gained from experiments towards automatic fold onset detection (see Section 6.4.2), strongly suggest that the model deformation alone will most likely not suffice here. In particular the *information gap* between the actual paper and the current belief – represented by the paper model deformation – is the issue. However, due to the nature of the proposed observation-based model control law (see Section 6.1.3) there are other promising model features, that a possible system could build on. By initially assuming simple and non-degenerate paper configurations, the localized magnitude of this information gap, i.e. the local modeling error, can directly be used to infer the position and orientation of a fold. To this end, the *model stress map*, which associates a so called stress value to each position of the paper surface is introduced. Assuming that, due to the bending stiffness of the physically constrained paper model, the modeling error is statistically worse in the local vicinity of a fold, an analysis of the distribution of high stress values is likely to allow for a good enough approximation of the fold geometry.

The Model Stress Map

Since the above motivated local modeling error is an unknown quantity, the model stress map is introduced as an approximating function $C_{\text{stress}}(\mathbf{x}^m) : P \rightarrow \mathbb{R}$ that associates a local *stress* value to each position of the paper surface. As a simplification, a stress value s_i is first defined for each of the paper nodes $n_i = (\mathbf{x}_i^m, \mathbf{x}_i^w)$ (see Section 6.1). The actual mapping function is then



(a) In-plane rotation (b) Soft bending along long edge (c) Hard bending along long edge

(d) Hard twisted bending (e) Creasing without added fold (f) Creasing after fold was added

Figure 6.29: Visualization of the generated model stress map C_{stress} (right part of each sub-figure) in a set of selected manipulation situations (left part of each sub-figure). The results of the spatial-PCA-based direction estimation is visualized as a red-cross on top of the stress map. The brighter white/grayish stress map values are the ones taken into account for the direction estimation. Values less than the threshold θ_C (tinted bluish colors) are disregarded. The actual value domain of C_{stress} depends on several internal parameters, such as the controller's proportional gain λ (see Section 6.1.3) and the number of controller iterations conducted per frame and is therefore not directly linked to a real physical quantity. For the sake of comparability, the gain's domains were equally quantified in all sub-figures.

defined by conducting Voronoi tessellation, i.e. by employing a simple nearest neighbor look-up along the paper surface: $C_{\text{stress}}(\mathbf{x}^m) = s_{\text{nn}(\mathbf{x}^m)}$, where $\text{nn}(\mathbf{x}^m) = \text{argmin}_i(|\mathbf{x}^m - \mathbf{x}_i^m|)$. The stress value s_i that is supposed to approximate the local modeling error at the model position $\mathbf{x}_i^m$ of node i, is computed by accumulating the velocity-based model updates for that node over all model control iterations corresponding to a single input frame (see Section 6.1.3). Initial experiments revealed that only the ICP-based velocity updates are suited for this. The surf-feature-based updates include too much noise and thus negatively affect the stress map quality. The accumulated ICP-based update velocities serve well for the approximation of the modeling error as it can be assumed that model nodes that are close to the observed corresponding point cloud points are moved only minimally. In contrast, nodes that can not be moved completely towards their point-cloud counter-parts, due to actual paper movements or due to the constraints, still receive an update move command from the observation-based update mechanism. While the discrepancies caused by movements or bending of the actual paper are implicitly exponentially decreased during the course of the update mechanism's iterations, deformations that can not be modeled due to the physical model constraints lead to a steady tension that reaches an equilibrium between external update forces and internal physical forces and thus result in continuously high update velocities. This assumption allows us to define three node classes, characterized by their stress value ranges:

1. Steady nodes that correspond to model regions that match the observation (s_i : low)
2. Nodes corresponding to slight bends or that are moved in space (s_i : medium)
3. Nodes that are close to strongly bent or folded paper regions (s_i : high)

Figure 6.29 shows a set of selected manipulation situations together with the resulting stress maps. In the case of a simple in-plane rotation (see Figure 6.29a), significant ICP-based updates are only generated for the border nodes of the model. By employing the default ICP-variant presented in Section 6.2.1, nodes that correspond to occluded parts of the model, such as those hidden by a

manipulating hand, are implicitly drawn towards the nearest non-occluded sections of the visible point-cloud. Due to the fact that these are commonly further away, this leads to larger update velocities and thus in higher stress map entries for the occluded parts. This effect can be seen in the center of the right edge of C_{stress} in Figure 6.29a(right). While the effect is only barely visible in the example, larger amounts of occlusions are likely to negatively influence the stress map's validity. In 6.29b, the paper is softly bent along the long edge. Due to the large bending radius, the bend appears blurred in C_{stress}. In contrast to this, in case of sharper bending (see Figure 6.29c,d), the fold is more distinct in the stress map. The fact that the position of the corresponding computed fold geometry estimate is some centimeters off to the left (see Figure 6.29c,d), could be explained by arguing that the lower part of the model overshoots the associated part of the real paper (see Figure 6.29c(left)). However, as a position error orthogonally to the fold estimate can be observed in the other examples as well, a different effect must be responsible for this. A deeper analysis (see Figure 6.30) revealed that the point-to-point association accuracy of the ICP-based update mechanism is much higher for the visible parts of the paper. A heuristical compensation mechanism is presented along with the description of the particle creation. Figure 6.29d(left) shows that even a slight twisting of the bend is recognized by the stress map. In case of a hard fold (see Figure 6.29e), the resulting stress map reveals a comparably wider structure with also a higher amplitude and the fold geometry estimation is very accurate. In contrast, after (manually) adding the fold line to the model (see Figure 6.29f(left)), the relaxation of the bending constraints along the fold allows the model to perfectly fit the observation and thus both, the fitting-error and stress-map entries become very small. Therefore the resulting stress map does not show any significant structures anymore (see Figure 6.29f(right)).

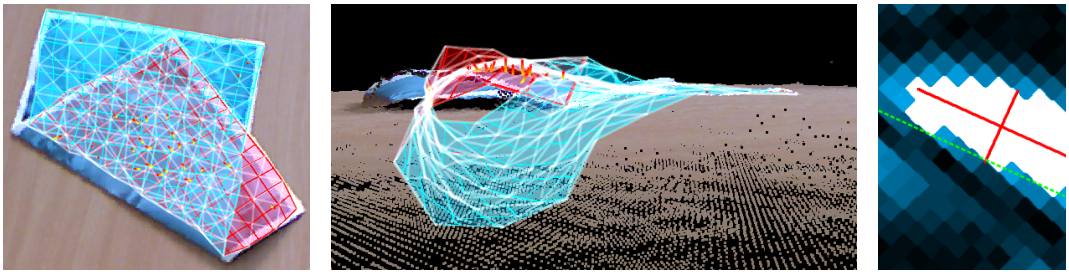
Extracting the Fold Geometry from the Stress Map

In order to compute the most-likely fold geometry, the stress map is initially thresholded using a dynamic threshold $\theta_C = \alpha \max(C_{\text{stress}})$. The relative threshold α, which was manually tuned to a value of 0.5, allows a corresponding fraction of low stress map values to be disregarded in the next step. Here, the center-of-gravity and the orientation of the axis with the highest variance of the remaining stress map entries is computed using a value-weighted spatial PCA-approach. The fold geometry is intrinsically represented by a vector $\mathbf{c}_{\text{est}} \in [0,1]^2$ where each of the two components describes one relative intersection position of the fold with the unrolled boundary of the paper surface.

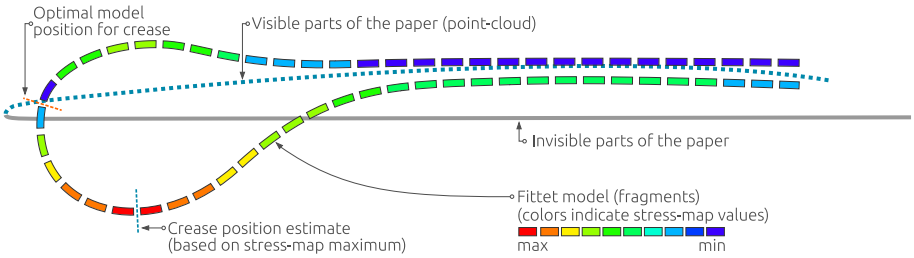
Particle Creation and Score Calculation

In situations in which the adding of a fold was detected or triggered, the particle system is initialized. To this end, the initial fold geometry guess $\mathbf{c}_{\text{est}}$ is reformulated, yielding the 2D center $\mathbf{c}$ of the fold and the orientation γ. For each spawned particle (except for the one that represents the no-fold hypothesis), $\mathbf{c}$ and γ are altered using Gaussian distributions. The orientation of the initial guess turned out to be quite accurate, so a zero-centered Gaussian with a small standard-deviation of 2 degrees suffices. As the orientation is altered separately, the position has to be altered along the orthogonal axis to the fold only[11]. In order to compensate the shift that was detected in the fold line estimation (see Figure 6.30), the center positions of the spawned particles not only statistically spread using a Gaussian-distribution with a standard-deviation of 2 cm, but also statically shifted by 3 cm along the same orthogonal axis. The fold line estimate splits the paper into two halves. The sign of the static shift is chosen, so that the resulting fold is moved towards the half

[11] We explicitly avoided the use of standard polar-coordinates for the representation in order to avoid an unwanted coupling of angle and position.



(a) Top view (b) Side profile view (c) stress map



(d) Schematic side profile view

Figure 6.30: Finding the reason for the shift in the stress map. **(a)** In the top view, the fitting of the model seems very accurate. Only the fact that the model does not actually *reach* the fold lines indicates fitting discrepancies. **(b)** The side view of the identical situation reveals the extent of the fitting error, which is caused by the model's physical bending stiffness. **(c)** In conjunction with the side view (b), the shift in the stress map becomes comprehensible. The actual fold is indicated by the dashed green line. **(d)** The schematic side view explains the effect in a qualitative manner.

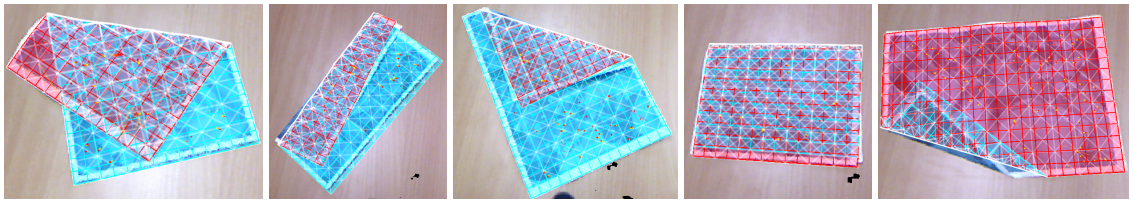
of the paper, whose vertex-mean is closer to the camera center. Examples and results are shown in the evaluation Section 6.4.4.
Once all particles are spawned, the system runs in its second operating mode (see Section 6.4.1) for a fixed number T of time-steps. After time step T, the system goes back to the first operation mode by replacing the last valid current model by the model that corresponds to the particle that performed best over the course of these time-steps.
The overall matching score S_i of a particle i is computed using recursive linear interpolation. Let $E_i[t]$ be the modeling error for particle i at time step t, then $S_i = S_i[T]$ is defined recursively:

$$S_i[t] = \lambda S_i[t-1] + (1-\lambda)\frac{1}{E_i[t]+\epsilon}\,,$$

where the initial score $S_i[0] = 0$ defines the basis for the recursion and a small ϵ avoids numerical issues in cases where $E_i[t]$ becomes too small. The formalism allows the scores resulting from more recent time steps to be made more important. The current error $E_i[t]$ is computed by a combination of four different components:

$$E_i[t] = F_i \cdot V_i[t] \cdot L_i[t] \cdot A_i[t]^2\,,$$

where F_i is the *fold penalty error*, $V_i[t]$ is the *node velocity error*, $L_i[t]$ is the *link tension error* and $A_i[t]$ is the *appearance error*. The different error terms are computed as follows. The fold penalty error, F_i, artificially increases the score of the non-fold particle. Assuming that w.l.o.g. particle 1 corresponds to the non-fold model, F_i is set to 1 for all i, except for $i = 1$, where F_i is set to 0.1. The node velocity error, $V_i[t]$, is defined by the average node velocity after the last iteration step in each cycle. It penalizes nodes that could, due to physical constraints, not be moved towards the observation. The link tension error takes into account that a good model's bending constraints



(a) (b) (c) (d) (e)

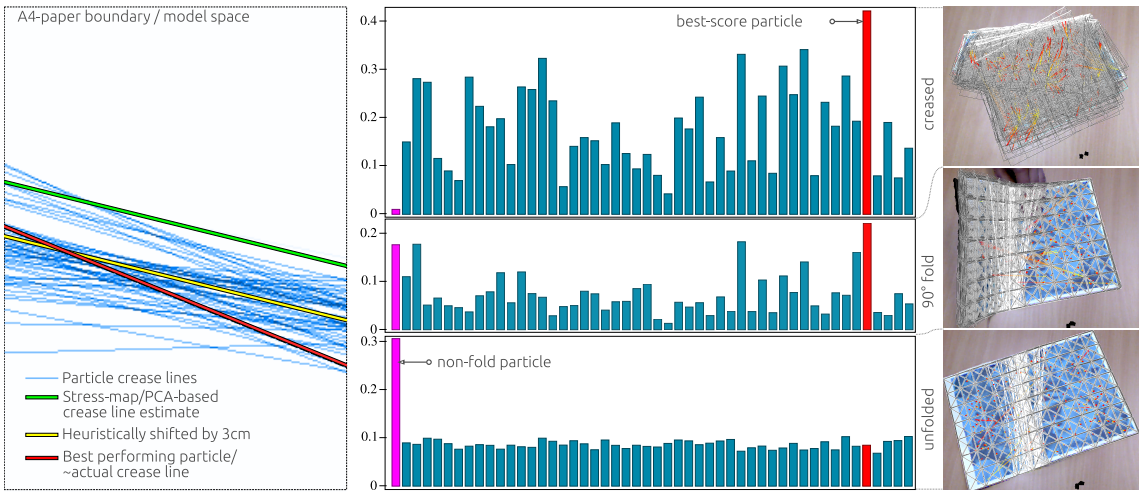
Figure 6.31: A set of exemplary fold geometry optimization results. The closer the fold gets to a corner of the paper, the less accurate the resulting fold estimate.

should be well satisfied. To this end, $L_i[t]$ is given by the mean difference between the constraints' desired distances and the actual distances of the linked nodes. The last component, the appearance error $A_i[t]$, describes how well a model covers the current paper point cloud segment. For this, the mean distance between the paper point cloud nodes and the closest corresponding model nodes is used. As the mean is formed over all paper point cloud points, the nearest neighbor search is internally sped up by inserting the model node positions into an Octree structure. The appearance error is included in a squared fashion to amplify its effect even if a large fraction of the point-cloud is covered perfectly.

6.4.4 Qualitative Evaluation

For the evaluation of the fold estimation system, several folding experiments were conducted. To this end, the real paper was arbitrarily folded by hand. After giving the system some seconds to adapt the fold-less model to the observation as well as possible, the adding of a fold line was manually triggered so that a pre-defined number of 50 particles were spawned. The particle count was manually adjusted to this value so that the resulting frame-rate drop to about 2 FPS[12] still allowed the model to be tracked satisfactorily (given that the manipulation is deliberately carried out slowly). The number of time-steps T the system stayed in the particle mode was set to 20, i.e. after about 10 seconds, the best particle was selected. Figure 6.31 shows a set of examples. It is apparent that folds that are closer to the corner of the paper are detected slightly less accurately. This can be explained by the fact that the computed stress map shows more edge-artifacts here and thus the PCA-based centroid computation becomes less robust. However, the overall accuracy can be said to be *very promising*. It is very likely that fold estimates given by humans after seeing the folded sheets of paper would be much worse. A more detailed examination of the example fold depicted in Figure 6.31a is given in Figure 6.32. Figure 6.32a visualizes the initial PCA-based fold line estimate (green line), as well as the heuristically shifted origin line used as the center for the particle distribution (yellow line). In addition, the random spread of the spawned particles is shown. In the example, the position estimate was comparably better than the angle estimate. The used Gaussian distribution to generate random fold positions and orientations in the vicinity of the origin covered that variance sufficiently, so that a convincing best particle (red line) could be found. Figure 6.32b compares the course of the temporally accumulated particle scores as the paper is unfolded after the particle creation. To this end, the automatic transition to the single-model mode after T time-steps was explicitly disabled. In the initial state (top plot), right after the particle creation, the variance of the particle performances is very high, but the best rated particle (highlighted red) is significantly ahead of the second winner. The non-fold particle (highlighted purple) logically performs worst although its score is, due to the F_i error

[12] Performed on an Intel®Xeon®E5-1620 CPU running at 3.6GHz with an NVIDIA®GeForce®GTX 660 Ti graphics card.



(a) (b) (c)

Figure 6.32: Visualization of the particle variances and performance scores. **(a)** Visualizes the fold lines that correspond the different particles. In addition, the green line indicates the PCA-based initial fold geometry estimate which is then heuristically shifted by 3 cm (yellow line). The fold line that corresponds the best performing particle is highlighted by the red line. **(b)** Shows the scores, S_i, of the 50 particles in three different folding situations. For this, the particles were created only once when the paper was folded and the particles were not dissolved after 20 iteration steps. Instead, all particles were continued to be tracked in parallel even when the fold was manually unfolded. **(c)** Shows corresponding folding situations with a wire-frame overlay of all 50 models.

component, already amplified by a factor of 10. Figure 6.32c(top) shows that the diversity of the models is also reflected by the corresponding model set wire-frame overlay image. In the second situation, the fold was partly undone, resulting in a 90 degree fold. Here, as shown in Figure 6.32c(middle) the models only differ significantly on one side of the paper, which is indicated by the now much smaller variance in the corresponding particle error plot. While the former best particle still achieves the highest rating, the non-fold particle is now within the best performing 10%. This is again a result of the particle's *fold penalty error*-term. Even though the combination of the other model and appearance-based error components is higher, its score is now comparable to score of the other particles. If, however, the paper is flattened out completely, as shown in Figure 6.32b,c(bottom), the system robustly favors the non-fold particle, whose score is now about three times as high as the best other particle's score.

6.4.5 Discussion

Even though a robust method for fold onset detection could not be found, the research towards the realization of such a system provides a valid scientific contribution. In addition, the course of the experiments yielded valuable insights into the particular difficulties that one is confronted with when trying to develop such a system. Moreover, the preliminary used manual fold onset triggering mechanism is likely to be sufficient for robotic paper manipulation.
The developed fold geometry estimation and particle-based fold geometry optimization system revealed many promising results. While using a most-simple particle system approach that could still be optimized along several directions, the introduced physics and appearance-based particle score seems to cover most of the important aspects already very well. The conducted experiments show that both, the stress map-based initial geometry estimation as well as the particle-based optimization perform astonishingly well for several different model-axis-aligned and also diagonal

folds.
After the development and the prototype implementation of the model stress map, the PCA-based fold geometry estimation method seemed to be a promising candidate that could also help to solve the original fold onset detection problem. However, it again turned out that such a system still did not work in many situations. In particular, a stress map, such as the one shown in Figure 6.30c could also be the result of occluding the center of the paper with one hand, so again, further heuristics become necessary to disambiguate such situations.
A further, so far not addressed issue of the presented system is the fact that it is not able to easily detect a second or any further fold. The main reason for this is the natural bending behavior of paper which is also reflected by the physical model. The surface distance preservation of paper means that bents and folds are not allowed to intersect. Excluding folds that are not parallel but that cross outside the bounded paper surface, the only exceptions are given by very hard folds and by folds that are fully undone. In the latter case, the paper theoretically behaves like an unfolded sheet of paper. In the case of a hard 180 degree fold, the resisting part of the paper surface shrinks into the singular intersection point of the folds. If a real sheet of paper is folded iteratively so that the folds intersect, a very small area around the fold intersection is locally stretched to make the fold possible. For a fold that was undone before folding across it, the behavior is different. Here, a minimal remaining fraction of the fold would theoretically not allow the crossing fold to be applied. The real paper nullifies this issue, by automatically straightening the area around a potential fold intersection. Since neither of these effects is explicitly modeled in the physics engine, the model becomes very stiff along each direction not parallel to an already added fold. Unfortunately the resulting increased stiffness even overcomes the perception-based model update mechanism and thus, the stress map, which is based on the idea that the model at least approximates a newly added fold be a smooth curve, does not yield enough information for the initial fold line estimate.
A large fraction of all these difficulties could be overcome by using the particle-system idea in a more general fashion. Instead of explicitly linking particle-creation and dissolution to certain events such as triggers and timeouts, it would be possible to permanently track the model by a set of particles, each representing a hypothesis about the current model configuration. The idea of particle-based tracking is actually not new. Already in 1998 Isard and Blake [1998] introduced the *CONDENSATION* method, whose performance was verified in several 2D-silhouette tracking tasks. In a similar fashion, a normalized particle score $\hat{S}_i = S_i / \sum_j S_j$, could be used as a re-sampling probability for consecutive particle generations, which would implicitly not only feature an exploration of the space of possible fold configurations, but it would also automatically perform iterative fold geometry optimization. The main reason why the potential of such a hierarchical system was not evaluated at least in a prototype implementation is the fact that the necessity to track many particles in parallel in real-time would require too much code re-organization and optimization.

6.5 Robotic Manipulation of Paper from a System Perspective

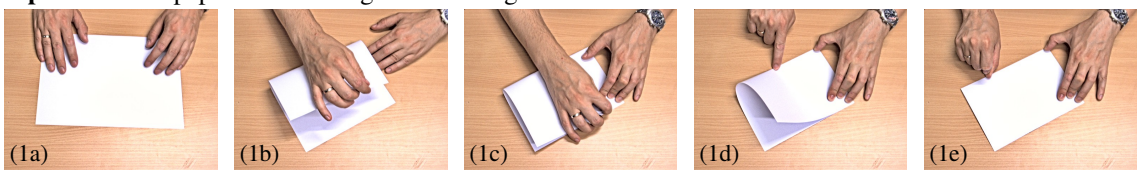
While the extensions along the modeling and the perception axes were accompanied by prototype implementations, extensions to the robotic control part are discussed in a purely theoretical and conceptional manner. Given the fact that we wish to discuss much more complicated interactive scenarios, it simply was not feasible to implement the ideas presented here on a real robotic system. Due to the complexity of the undertaken tasks, the actual implementations of the bi-manual paper manipulation systems for picking-up (see Chapter 4) and folding (see Chapter 5) paper were carried out in a more or less hand-crafted manner. In the following, an abstract framework for dexterous robotic paper manipulation is suggested in order to provide a structure for more general

and more powerful robotic systems. Once again, we do not claim to be able to come up with a fully fledged and ready-to-use robot control and planning system for such tasks. Instead, the following elaboration is intended to provide an initial idea of how the creation of a generic robot system for dexterous robotic paper manipulation could be approached.

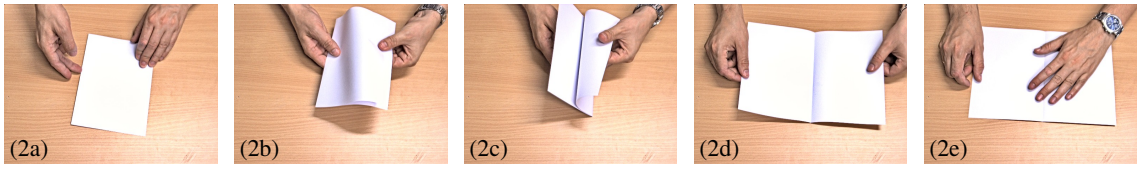
6.5.1 Bootstrapping a Bottom-Up Approach

To cope with the inherent complexity of our goal, a system for bi-manual dexterous robotic paper manipulation must be split into several sub-components. Once again, we suggest an overarching conceptual modularization into *perception*, *modeling* and *robot control* (see Figure 2.4), but this time, we assume the first two components to be given, in order to fully focus on the robot control part.
For a bottom-up approach, a first step is to select a set of minimally parametrized and atomic basic action primitives (BAPs) that serve as the basis for carrying out robotic paper manipulation. In order to bootstrap the bottom-up approach, a manipulation sequence of folding a simple paper aeroplane is used to create an initial minimal, yet sufficient, set of primitives for the given task. The set can be extended or adapted later in order to support other interactions, which might require additional or more general BAPs. Before these primitives can be derived, a description of how to fold an A4 paper aeroplane is provided as a basis for further explanations (see Figure 6.33). Please note that the written instruction steps in this description are optimized for an adult who has a good knowledge of paper manipulation already and therefore is able to understand and implement higher-level steps. Even though the written instructions would suffice for most people to easily follow to build the desired paper aeroplane, five images are provided along with each each step to simplify the following discussion. The fact that the written instructions assume a lot of information to be either known or to be deduced by the reader employing both context and real world knowledge, underlines the difficulty of the task.
In the following discussion, we will successively work through these steps and discuss how they could be accomplished with an anthropomorphic robot system. To this end, the steps are subdivided into their smallest logical units, whose functionalities are formalized as BAPs and thereby retained for later use. New BAPs are only created if a part of a processed instruction step cannot be realized using the existing set of BAPs. As new BAPs are added to accomplish the current task, suggestions of how they may be parametrized to achieve more varied outcomes are often presented. In addition, in order to ensure we arrive at a minimal set of BAPs, the definition of a new primitive is sometimes avoided by extending or generalizing an existing one. To verify our assumption that the preferred re-use of BAPs leads to a convergence over time as more and more different manipulation actions are included, we subsequently continue the process for a set of further more general manipulation examples (see Section 6.5.4). For some of the introduced primitives, additional figures are provided that illustrate details that are not optimally visible in top-view of the sequence Figure 6.33.

Step 1: Fold the paper in half along the short edge and harden the crease.

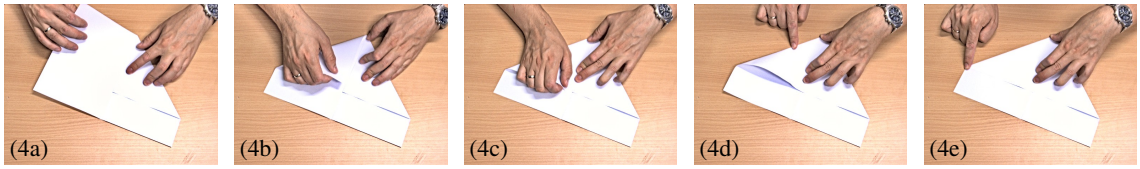


Step 2: Undo the previous fold.

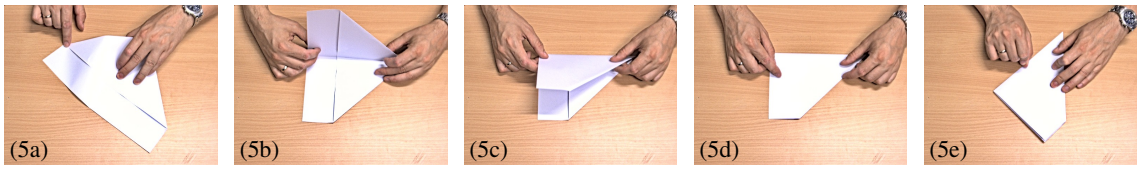


Step 3: Diagonally fold one corner towards the center fold and harden the crease.

Step 4: Repeat with the opposite side – mirrored along the initial fold.

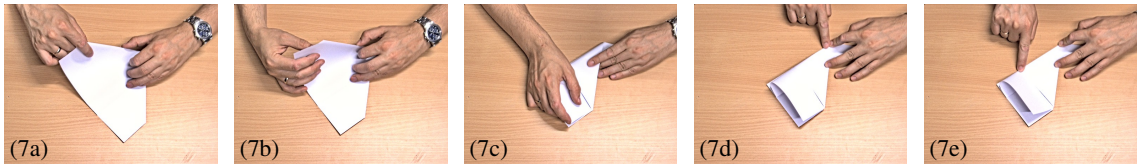


Step 5: Redo the first/center fold so that the diagonally folded edges are on the inside.



Step 6: Fold the top two layers parallel to and at a distance of about 4cm form the first/center fold.

Step 7: Turn the paper around and mirror the previous operation on the other two layers.



Step 8: Adjust the last two folds so that the wings are at 90° to the fuselage..

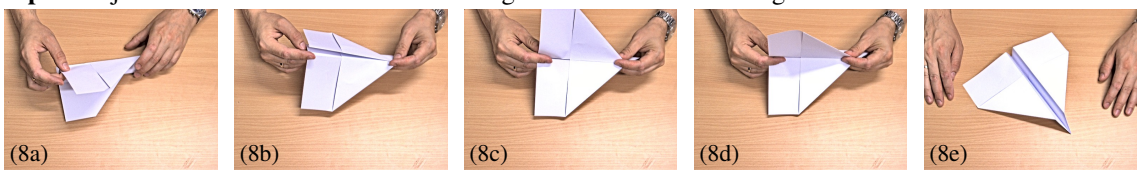
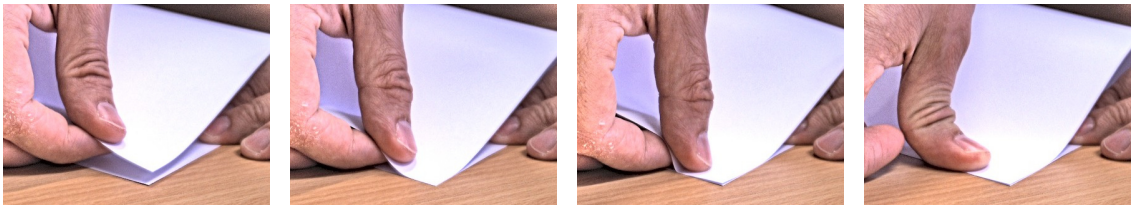


Figure 6.33: Instruction steps to build a simple A4 paper aeroplane.



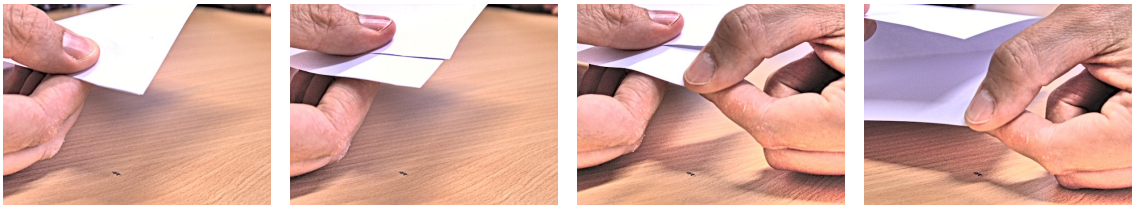
(a) Corner alignment (b) Fixate with both fingers (c) Shift pressure to thumb (d) Release forefinger

Figure 6.34: Explanation of the *PlaceFixate* primitive. **(a)** The top layer of the paper is pinch-grasped to perform the desired alignment. **(b)** Both thumb and forefinger are used to fixate the layers of the paper. To this end both the back of the forefinger and the tip of the thumb must be horizontally aligned by slightly *rotating* the maintained grasp. **(c)** Once the tip of the thumb fixates both layers, the grasp and the downwards pressure exerted by the forefinger can be released. **(d)** The forefinger is withdrawn from between the layers of the paper to allow the paper edge to relax into a straightened configuration.

Step 1: Fold the paper in half along the short edge and harden the crease

(Introduces primitives: *RigidMove*, *Fixate*, *Grasp*, *Move*, *Release*, *Align*, *PlaceFixate* and *Swipe*)

As Step 1 is identical to the robotic folding experiment described in Chapter 5 we directly use the insights gained there. In order to optimize the interaction space of the robot hardware and to enable the robot to reach certain parts of the paper, the paper needs to be rigidly moved during the manipulation. This leads to the definition of the *RigidMove* primitive. In addition to the paper shifting system presented in Section 2.2, common rigid-body pick-and-place frameworks can be employed here. The *RigidMove* primitive uses the target pose as a parameter and thus has to autonomously decide whether to shift or to first pick and then place the object again. The next BAP needed for Step 1 is *Fixate*, which means that the paper is fixated at a certain paper space coordinate. Usually, this can be achieved by pressing one or several layers of the paper against the work-top, but certain configurations might require a part of the model to be fixated without exploiting the work-top by using a pinch-grasp. *Fixate* needs to be performed with at least two fixation points to increase the leverage effect against an unwanted rotation of the paper. For the initial folding of the paper in half, *Fixate* is performed on one half of the paper while it is grasped and bent by the other hand. The next three primitives, *Grasp*, *Move* and *Release*, are used to actually manipulate the paper. *Grasp* is parametrized with the paper space coordinates, the fingers that are used and the direction of robot hand relative to the paper. When initially sequencing manipulation steps, *Grasp* must often be prepared by an explicit *RigidMove* step to expose the to-be-grasped part of the paper appropriately. The *Move* primitive, parametrized by a pre-planned movement trajectory that is defined relatively to the fixated part of the paper, is followed by a specialized BAP, *Align*, which can align paper corners, edges or creases. Due to the high precision that is needed to avoid an increasing inaccuracy while carrying out successive manipulation tasks, *Align* should be implemented in a closed-loop fashion. For the case of Step 1 of the paper aeroplane example, *Align* is used to align the corner of the bent-over upper paper layer with the corner of the fixated bottom layer. The used point-to-point alignment is comparably simple as the alignment has no intrinsic null space. In contrast, e.g. diagonal point-to-line alignments require additional constraints to be defined in order to become fully specified. After accomplishing the desired alignment, a very special dexterous ability is needed. The grasp that was used for the alignment is directly reused to fixate the aligned paper layers. This actually commonly used by people technique is achieved by aligning a pinch-grasp in such a way with the paper on the work-top, that both the thumb and the forefinger have direct contact the work-top (see Figure 6.34). By increasing the vertical contact force of the thumb, the paper gets fixated by the thumb's tip, so that the opposing forefinger can be slid out. This placing/fixating movement defines the *PlaceFixate*



(a) Perfectly aligned layers (b) Shift to displace layers (c) Grasp isolated layer (d) Layers are separated

Figure 6.35: Explanation of the *PinchDisplace* primitive. **(a)** The sheet of paper is folded in half leading to a perfect alignment of two folded layers. **(b)** By shifting the thumb along the tip of the opposing forefinger while maintaining the pinch-grasp force, the two edges are displaced. **(c)** Now, the isolated bottom layer can be pinch-grasped by the other hand. **(d)** Once, only the bottom layer is grasped, the layers can be fully separated.

primitive, which is later needed again. *PlaceFixate* allows the other hand that was originally used to fixate the bottom layer while *Align* was active, to re-position in order to fixate both aligned layers. Internally, this requires a *Release* primitive to be concatenated with another *Fixate* step. *Release* not only implements a *careful* releasing of the initially fixated part of the paper, but also automatically moves the hand to a free position on a trajectory that does not intersect with the upper layer of the paper. The hand can now fixate both layers (see Figure 6.33-1c), freeing up the other hand to perform the *Swipe* primitive to actually crease the paper. As demonstrated in Chapter 5, creasing can be carried out in a two-step fashion. In the first run, minimal pressure and four fingers are used to slightly crease the paper along the center fold. After that, the fixating hand is re-positioned closer to the fold line (*Release* followed by *Fixate*) to ensure that the paper is not accidentally moved during the following higher-force/one-finger swipe movement. A human can accomplish such a hard crease by performing a single swipe movement (see Figure 6.33-1d), but here a very complex coordination of both hands is needed, which is even more difficult for a robotic system to accomplish than the suggested two-step approach.

Step 2: Undo the previous fold

(Introduces primitives: *PinchDisplace* and *Stretch*)

Step 2 directly confronts us with a thus far unsolved task. In order to undo the previous fold, the now perfectly aligned layers of paper have to be separated. This is achieved by pinch-grasping the paper at a position close to a corner of the opposite side of the crease. Once again, this is done by positioning the paper using *RigidMove* followed by *Grasp*. Now, after lifting the sheet of paper, a complex in-hand movement must be performed to allow the other hand to grasp only one of aligned layers of the paper (see Figure 6.35). To this end, the pinch-grasping thumb is moved along the surface of the opposing forefinger tip towards the forefinger's nail, which will be referenced as the *PinchDisplace* primitive. Due to the fingertip friction, the corners of the two grasped paper layers are displaced, exposing the layer that is touched by the forefinger, which allows the other hand to pinch-grasp only that layer. Releasing the pinch-grasp that was used to perform the *PinchDisplace* movement leads to a further unfolding of the paper (see Figure 6.33-2c).

In order to put the paper, which now contains a center fold line parallel to its short edge, flat onto the worktop, it must first be spread. To this end, the opposite corner to that which is being held, is grasped and both corners are put simultaneously on the work-top, while maintaining a stretching force along the diagonal of the paper. Even though, this move could be formalized by a two-handed combination of the *Grasp* and *Move* primitives, we introduce it as a special primitive, *Stretch*, to emphasize the complexity that is caused by the difficult coordination of both hands.

The maintaining of a stretching-force is actually a very difficult task to achieve as common finger-tip touch sensors still have difficulties measuring tangential forces. The stretching is followed by a flattening of the sheet of paper in order to re-adjust the center folds resting angle to be $180°$. This is achieved by finishing the *Stretch* primitive for one hand using *PlaceFixate*, followed by performing a low-force four-finger *Swipe*-movement along the fold.

Step 3: Diagonally fold one corner towards the center fold and harden the crease/ Step 4: Repeat with the opposite side – mirrored along the initial fold

(no new primitives introduced)

Creating the diagonal folds needed for the front-wings of the paper aeroplane in Steps 3 and 4 is similar to the creation of the initial center fold except for the parameters of the *Align* primitive. Due to the fact that the fold is diagonal, a line-to-line alignment (paper edge to the center fold line) is needed. For a human, this is particularly complex, as an exact alignment can only be achieved if the top to-be-aligned paper edge is straightened during this step. However, as we initially assumed accurate perception and modeling components to exist, the exact target point for the bent-over paper corner on the center fold line can be derived from the model, transferring the task into a fully specified point-to-point alignment problem that can be solved similarly to the initial center fold employing the *PlaceFixate* and the *Swipe* primitives. The opposite diagonal wing-fold can be carried out similarly but mirrored along the paper's center fold axis as the target-point for the paper corners are identical.

Step 5: Redo the first/center fold so that the diagonally folded edges are on the inside

(no new primitives introduced)

The re-creation of the center fold in Step 5 can be achieved analogously to the initial fold (Step 1), but this time, the already existing fold-line leads to an automatic alignment of the folded layers. Similar to a human who folds a paper aeroplane, the system would here have to decide whether to *trust* the accuracy of the previous folding steps by simply assuming the hinge-joint like behavior of the center fold to be correct or by re-ensuring the correctness of the center fold by explicitly employing the *Align* primitive again. For the generated primitive sequence this means that *Align* can either be used or simply be left out. In an optimal case, the system would test the necessity of using *Align* by checking the perceived alignment error.

Step 6: Fold the top two layers parallel to and at a distance of about 4cm form the first/center fold

(Introduces the *PinchFold*-primitive and generalizes the *Align*-primitive)

At the beginning of Step 6, the *PinchDisplace* primitive might be needed to separate the two folded wings. Depending on the hardness of the center crease, the wing-tips could also be far enough apart from each other, allowing the top wing to simply be pinch-grasped. Both approaches would allow the system to use the *Grasp* primitive with both hands to achieve a hand/object configuration similar to Figure 6.33-6a. However, the subsequent bi-manual folding technique requires a thus far unseen dexterity and coordination skill of both hands, leading to the definition of the *PinchFold* primitive (see Figure 6.36). The starting configuration of this primitive is that two pinch-grasps are placed closely and in parallel to the desired fold-line (see Figure 6.33-6a and 6.36a). Then, the two pinch-grasps are used to bend over the grasped layer of the paper (see Figure 6.36b,c). To this end, the forefingers are used as rotation centers while they fixate the bottom layer of the paper

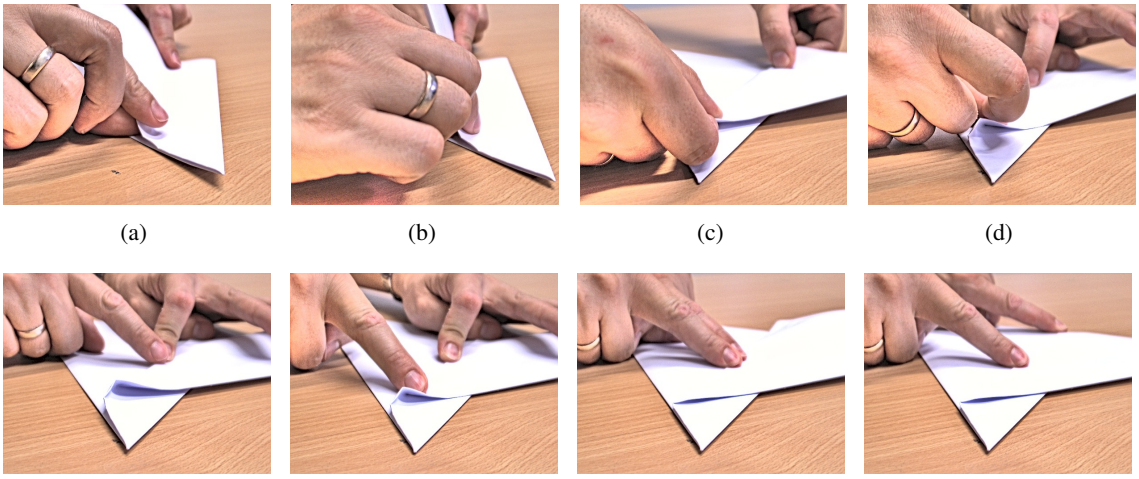


Figure 6.36: Manipulation sub-steps to get from Figure 6.336a to Figure 6.336c leading to the definition of the *PinchFold* primitive. **(a-c)** Bi-manual folding using two pinch-grasps. **(d)** Dexterous fixate using the left forefinger while maintaining *Fixate* with both thumbs. **(e,f)** *Swipe* primitive carried out with the right forefinger tip to harden the crease. **(g,h)** *Swipe* primitive with the left forefinger for the other part of the crease.

on the tabletop. In order to move the thumb-tips along the surface of the fixating forefingers the whole hand-posture must be controlled (see Figure 6.36a-c). At the end of this bi-manual rotation movement (see Figure 6.36c), the fold-alignment is controlled by slight x/y position-changes of the thumbs relative to each other, requiring a severe generalization of the *Align* primitive. For the alignment, the parallelism of the current fold and the existing center fold (bottom of the aeroplane's fuselage) as well as the parallelism of the aeroplane's rear paper edges must be measured and optimized. As soon as the thumb tips establish contact (see Figure 6.36c), an extremely dexterous maneuver is needed to withdraw the forefingers from between the folded layers (see Figure 6.36d). The right forefinger is used to fixate the fold while it is still held by the two thumbs. Only when the fold is fully fixated by the left thumb and forefinger (see Figure 6.36d,e), is the right hand free to perform a one finger *Swipe* primitive to harden the crease. For the other side of the crease, the right hand is used for *Fixate* while the left hand carries out the *Swipe* primitive (see Figure 6.36g,h).

Step 7: Turn the paper around and mirror the previous operation on the other two layers

(no new primitives introduced)

The folding of the other wing of the paper aeroplane in Step 7 can be carried out analogously to Step 6, except for the fold alignment sub step. In contrast to the first folded wing, for which only an approximate distance of 4cm to the center fold was required, the fold alignment in Step 7 also needs to achieve an ideal alignment of the two wing folds. Actually, as we have to allow for the system making a small but not completely negligible error when applying a fold, the over-determination of the exact fold geometry here leads to a trade-off between the different to-be-aligned edges and folds. By employing world knowledge about the aerodynamics of paper aeroplanes, a human expert would place high importance on the symmetry of the wing folds.

Step 8: Adjust the last two folds so that the wings are at 90° to the fuselage.
(no new primitives introduced)

The final abduction of the wings (Step 8) can be achieved in many different ways. While the very special bi-manual 3-finger fold adjustment technique presented in Figure 6.33-8c,d is quite natural for a human, a robotic system could use a simpler manipulation sequence that can be created from the existing primitives. As a matter of fact, the presented *PinchFold* primitive can be employed inversely to this end. Thus, by using a combination of *RigidMove*, *Grasp* and *PinchDisplace*, the robot could separate the wings in order to achieve an appropriate initial hand/paper configuration for *PinchFold*. The actual adjustment of the fold angle (here from about $180°$ to $90°$) is typically accomplished by an explicit *over bending* action, that is, we adjust the fold to about $60°$ so that the paper's plastic resiliency leads to an actual folding angle of the desired $90°$. The needed angle depends on many variables such as the paper's thickness and stiffness, the reduction of that stiffness along the fold caused by the creasing and also on the grasp-force that is used during the inverted *PinchFold*. In addition, a multiple execution of the same primitive will yield different results. The physics-based simulation might prove useful to acquire a well suited initial guess of the angle, but when performing the abduction, a closed-loop trial and error based execution might be necessary.

6.5.2 An Extendable Set of Basic Action Primitives

The analysis of the folding sequence to make a paper aeroplane (see Section 6.5.1) provided us with an initial set of BAPs. Before we try to realize other similar and also very different manipulation actions with paper and paper-like objects (see Section 6.5.4), the existing primitives are described more systematically. Some of these primitives need to be implemented as combined controllers that must achieve or maintain a set of different sub-goals, which is commonly done using a *control basis* approach [Huber and Grupen, 1997]. In a control basis framework, high and low-level controllers can be composed in a hierarchical fashion using the *subject to*-relation. If a controller C_1 is executed *subject to* a controller C_2 (commonly written as $C_1 \lhd C_2$), the framework will internally execute C_1 in the null-space of the potential-function that underlies C_2. The potential function associated with the controller can be derived either from a high-level state logic or sequence planner or from a low-level closed-loop feedback mechanism.
The suggested BAPs are, however, defined on a level above the controller concept of the control basis framework. Therefore, BAPs can be single controllers or combinations of controllers that have to be executed in a dependent fashion using the *subject to* relation. In addition to this, there are situations in which BAPs themselves must be cascaded so that one BAP is executed *subject to* another BAP.
In the following, the presented BAPs are discussed with regard to common use-cases, parameters and other related BAPs. In addition, possible difficulties and extensions for an actual implementation are suggested. It is important to underline that we do not claim to provide blueprints that allow the primitives to be implemented directly. Instead, this section aims to provide information and useful insights and to highlight predictable problems in order to bootstrap and support the development process of such blueprints. Sometimes we suggest the extension of a primitive's desired functionality to automatically perform other actions before, in parallel or after the execution of the basis function of the primitive. It is obvious that moving a certain action from *to be performed before the primitive* to *an initial part of the primitive* does not significantly facilitate the actual development of that functionality. Instead, including certain actions into primitives makes these more powerful and easier to integrate into an interaction sequence and thus simplifies the global planning and primitive sequencing mechanism.

Align: The *Align* primitive is employed to achieve an alignment of paper edges, corners and fold lines. To this end, *Align* is usually executed while the part of the paper, to which the grasped part needs to be aligned with, is fixated using the *Fixate* primitive. A high accuracy can be achieved by implementing *Align* using a closed-loop controller. The desired alignment goal can be defined as a single point-to-point alignment or it can encompass several to-be-aligned geometrical primitives and thus, the target might be over or under-specified. While in the case of an over-specification, an appropriate mediation function must be selected, an under-specification is usually complemented by additional constraints that could originate from a parent controller so that *Align* is executed *subject to* it.
It is conceivable that the low-level alignment is implemented purely on the basis of the information provided by the current model. However, a direct image-feature-based alignment method might prove more accurate in given situations.

Fixate: The fixation of certain parts of the manipulated paper is a very common and frequently used action primitive. If possible, *Fixate* is applied by wedging in one or several folded layers of the paper between the fingers and the worktop. Humans would usually prefer this technique as it is energy efficient and due to the fact that its friction-based contact to a steady worktop offers high stability. For a robotic system both of theses features might not directly apply, which could lead to differences in the final way a robot interacts with paper.
Internally, *Fixate* has two states. The first state is active to establish contact with the desired fixation point commonly employing both visual and tactile feedback to achieve this. After contact establishment, the second state is activated, in which *Fixate* maintains a given contact force. Active-posture based controllers for this skill were presented in the robotic folding experiment (see Section 5.5.3). The *Fixate* primitive is parametrized by a number of paper-space coordinates that are to be fixated. The coordinates have to be derived from the higher level action planner and sequencer as usually only the actions that are performed in parallel to *Fixate* provide the necessary information enabling optimal fixation points to be computed.
While most of the time, fixate can be performed with the fingers of an otherwise *free* hand, certain actions such the left forefinger fixation depicted in Figure 6.36c-f can require extremely dexterous movements, in which *Fixate* is applied with one finger of a hand while the other fingers are used to run a different action primitive. Thus, it can be necessary to execute *Fixate subject to* another BAP or even *subject to* another instance of *Fixate*.

Grasp: The *Grasp* primitive is used to either fixate (see the *Fixate* primitive) or move a part of (see the *Move* primitive) or the whole sheet of paper (see the *RigidMove* primitive). Grasp needs to be initialized with a given grasp-type, the to-be-grasped paper-space coordinate as well as the direction of the grasp. Internally, *Grasp* must be able to avoid collisions with other parts of the paper.

Move: *Move* is used for coarse paper deformation operations that can be planned in advance. A pre-planned relative movement trajectory is used as a preparation for another, more precise, manipulation action such as *Align* or *PinchDisplace* – usually while the other hand executes *Fixate*. Thus, the definition of the *Move*-primitive basically acts like a BAP-*wrapper* for an already well understood robot controller that works in task or object-space.

PinchDisplace: As demonstrated in Section 6.5.1 and in particular in Figure 6.35, a special ability is needed to separate aligned layers of paper. A simple implementation of the *PinchDisplace* primitive would use a pre-defined relative movement trajectory of the thumb with respect to the opposing forefinger. However, more elaborate variants also seem possible that use visuo-haptic feedback to optimize the grasp's contact force and visually ensure that the conducted relative finger movement actually yields the desired result. *PinchDisplace* can only work if the actual friction between fingertips and paper is higher than the friction between the to-be-separated paper layers. Due to possible non-linearities caused by the materials and construction of the robotic fingertips an

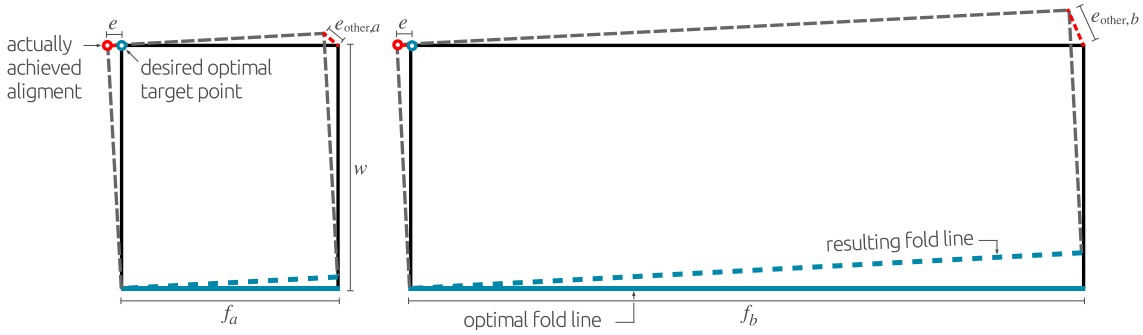


Figure 6.37: Additional explanation for the *PlaceFixate*-primitive, illustrating the expected alignment error, e_{other}, with respect to the given ratio between the fold length f and the orthogonal paper width w. **(left)** The quotient $f_a/w \sim 1$ results in an expected alignment error, $e_{\text{other},a} \sim e$. **(right)** Even though the point-to-point alignment error, e, is identical to the one in the left figure, the larger quotient $f_b/w > 2$, yields a proportionally larger expected alignment error $e_{\text{other},b} \gg e$.

optimal force can not necessarily be given in advance. Instead, a closed-loop controller based on high-frequency tactile feedback [Schöpfer et al., 2010] could be employed to maintain a minimal, yet sufficient, contact force. In our robotic folding experiment (see Section 5.5), we counteracted slippage by using rubber finger covers to increase friction.

PinchFold: The thus far most specialized of the defined BAPs is *PinchFold*, which is used for precise folding operations. While for the initial folding in half of the paper in our paper aeroplane example, a folding technique that consists of the steps *Fixate*, *Grasp*, *Move* and *Align* (FGMA) is employed (see Figure 6.33-1a-e), the later folding of the aeroplane's fuselage is achieved using the technique that led to the introduction of the *PinchFold*-primitive (see Figure 6.33-6a-e). Reviewing several random origami folding videos on the internet provided us with the insight that the FGMA-fold seems to be used only in cases, in which the desired target fold line divides the paper or a large-enough and not too-elongated part of the folded paper in half. Some people even seem to prefer a technique similar to *PinchFold* all of the time. There are several obvious reasons that explain this observation. In the case of the FGMA-fold, the folded part of the paper is usually grasped close to one end of the fold, which significantly facilitates the subsequent alignment step. This makes particularly sense if the paper is folded in half as here, the alignment is sufficiently specified by the alignment of the two corners and thus high accuracy can be attained. In contrast to this, if the fold geometry of the desired fold precision requires a second point to be aligned, the *PinchFold*-technique is often preferred. The latter also becomes mandatory for cases in which the length, f, of the fold is multiples higher than the width, w, (orthogonally to the fold) of the to-be-folded part of the paper. Given an expected point-to-point alignment error, e, (parallel to the fold). The expected alignment error, e_{other}, for the opposite corner is proportional to the quotient f/w (see Figure 6.37).

PlaceFixate: As explained in Figure 6.34, the *PlaceFixate* primitive is used as a natural and efficient transition between placing a part of the paper and immediately fixating it. Similar to the chosen example, it is commonly preceded by an *Align* step. Depending on the actual implementation, it might even be necessary to execute the initial *placing*-part of *PlaceFixate subject to* a simultaneously executed *Align* primitive. Otherwise a previously achieved perfect alignment might be undone during the fixation movement. Once contact is established so that the desired parts of the paper are fixated, the previously described thumb-rotation frees the forefinger, which can then be withdrawn in order to directly perform a subsequent BAP or at least to decrease the limitation of the fixating hand's possible interaction space. As *PlaceFixate* is assumed to be executed in a situation in which a part of the paper is already pinch-grasped, a parametrization that

allows a possible parent *Align* primitive to be selected, suffices here.

Release: Careful releasing of a previously grasped or fixated part of the paper, followed by an automatic withdrawal that avoids collision between hands and the paper is needed several times in many interaction sequences. While for a simple rigid placing of a folded or unfolded sheet of paper it would be sufficient to simply *let the paper fall* from above the target position, releasing the paper during an interaction sequence must not result in the paper accidentally displaced. The importance of a sophisticated *Release*-primitive was even shown with regard to our very early paper shifting experiment (see Section 2.2). By extending the functionality of the primitive to accomplish a full collision-free withdrawal of the robot hand, the primitive becomes even more powerful. To this end, however, an actual implementation must initially evaluate the whole scene, including the posture of both arms and hands and the complete deformation of the paper. How to actually perform the release-movement strongly depends on the used robot hardware. We exploited the compliant spring-like construction of the Shadow dexterous robot hands as a hardware buffering mechanism to maintain a given contact force while avoiding damage to the hands. However, the same spring-like behavior in turn automatically leads to an unwanted displacement during a release motion as the fingers tend to bounce back to their specialized postures. Other types of robot hands that implement compliance though a software-controller could natively suppress or even completely avoid this behavior.

RigidMove: The *RigidMove* primitive is used whenever the to-be-manipulated object pose needs to be altered. An actual implementation could use an existing pick-and-place framework. However, the non-rigid nature of the paper introduces a new layer of complexity to the trajectory planning. The main reason for this is the object's tendency to deform under the influence of gravity or even due to its own inertia. Given a desired target pose parameter, the internal implementation could employ physical simulation to find a suited trajectory.

Stretch: The stretching of a (previously folded) sheet of paper, which is formalized by the *Stretch*-primitive is an example of a bi-manual manipulation. In contrast to most of the other BAPs, which are also executed with two active hands, *Stretch* employs both hands in an equal but counteracting fashion. As mentioned earlier, a stretching force that emerges tangential to the fingertip surfaces is difficult to measure with existing tactile sensors, necessitating the integration of force torque sensors into the robot arms. Alternatively, an accurate perception module could be employed to estimate the stretching state of the paper. The definition of stretching on the basis of the shape rather than on the basis of the force would be an even more direct quantity that could be used. A possible implementation could use such a measurement function as a plugin-parameter or even employ several sources at once to perform sensor-fusion for more robust results. The *Stretch* primitive requires the paper to be grasped at two positions, so that the *to-be-stretched-out* part of the paper is located in the middle. In our example, *Stretch* did not contain either the initial two-point grasping of the paper nor the subsequent releasing of the paper. A possible generalization of the *Stretch* concept could require the primitive to be able to automatically compute and perform these actions, which would facilitate its integration into an action sequencing mechanism.

Swipe: The most simple imaginable variant of the *Swipe*-primitive could be configured with a linear paper space trajectory, defined by start and end points, the fingers that should be used and a desired contact force. However, similar to the aeroplane example, such a swiping motion is, due to the friction of the finger-tips, prone to move or rotate the paper, which is why the paper must be fixated appropriately by the other hand. While an unwanted translation of the paper can be satisfactorily prevented using an arbitrary *stable-enough* single point fixation, avoiding accidental rotation requires two points to be fixated that are optimally not too close to each other. In addition, at least one of the fixation points must be close enough to the starting point of the swipe-trajectory as otherwise the leverage effect can cause the paper to curl. In order to relieve the global planning and primitive sequencing mechanism, the definition of *Swipe* could be extended so that it that can

also automatically compute optimal fixation points.

6.5.3 Primitive Sequencing and Planning and Learning

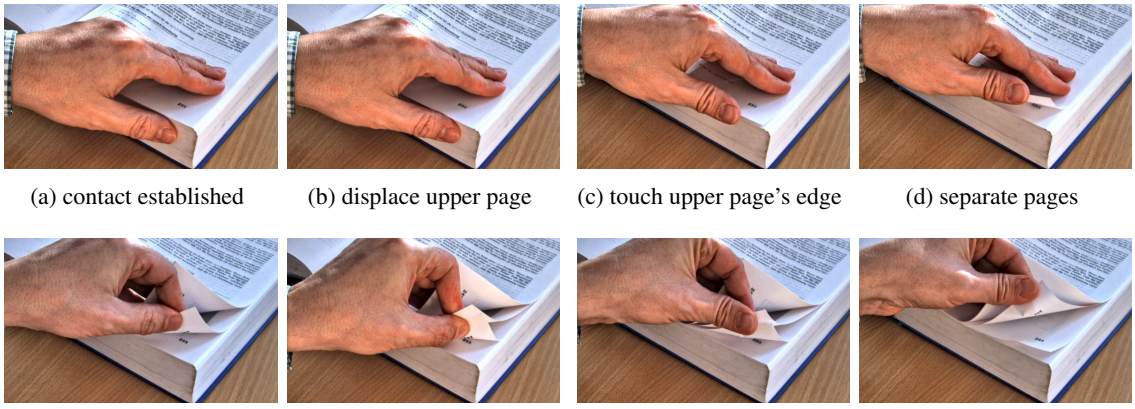
In the context of the introduced primitives, *learning* could be understood in different ways. From a conceptual point of view, a simple way to start would be to search for optimal primitive parameters. Such a learning process could be based on human demonstration or on an exploratory search. However, while the manual creation of training data is a very time-consuming task, exploration demands that either a real robot is used and thus leading to high hardware maintenance costs or that a physics simulation is employed, which might, due to an incomplete modeling of physical effects, yield sub-optimal or even wrong parameters for the real-world.
In contrast to this, learning could also be understood as the ability to find out how to plan and automatically sequence the existing primitives to achieve desired paper configurations. The two-fold nature of this problem can be underlined by splitting it into two sub-tasks:

1. How must the paper be moved/transformed to achieve the desired target deformation?
2. What robot movements are needed to make the paper move that way?

The first sub-task is mainly related to modeling and manipulating the paper. In most of the related work (see Section 5.1), and in particular when dealing with the formalization of origami folding, folds are modeled as temporally discrete events that happen between two consecutive time steps. In contrast, for the planning of robotic manipulation sequences, the dynamic nature of the folding process must be incorporated. Furthermore, many related systems model to-be-manipulated paper or cardboard objects as rigid faces connected by a finite set of revolute joints (analogous to a robotic kinematic chain), which allows classical robot-planning algorithms to be directly applied. While this is a sufficient approximation for the initial inference of folding sequences, the actual non-idealized folding of paper requires the deformability of the faces to be taken into account and these should therefore be reflected in the model. The first sub-task could be approached by extending an existing system by directly integrating a more advanced model, such as the one proposed in Section 6.1. As an alternative to this, a two-tier framework in which an initial traditional solution is refined with the addition of such a model seems also plausible.
The second sub-task of planning robot movements by sequencing and parametrized action primitives in order to achieve a desired paper-deformation trajectory is even more complex. This step is commonly simplified by using simple robot hands [Tanaka et al., 2007] [Liu and Dai, 2003] or even specialized folding robots [Balkcom, 2004; Lu and Akella, 2000] and accordingly, rather simple action primitives can be applied. In contrast, our vision is to use fully-fledged anthropomorphic robot hands and correspondingly sophisticated action primitives, both having several tens of degrees of freedom. Due to the resulting combinatorial explosion, such a problem presents classical planning approaches, such as PRM-based methods [Kavraki et al., 1996], with a very difficult task.

6.5.4 Using our Primitives to Manipulate other Deformable Objects

As a final thought experiment, we discuss, how well our developed set of BAPs is theoretically sufficient to carry out manipulation actions with other more general deformable objects. The selected exemplary set of actions extends from manipulations that are very similar to our paper folding experiments to radically different sequences. Our initial focus remains on planar (2D) objects. By restricting ourselves to 2D objects, we can achieve a much higher re-usability of the existing BAPs and thus limit the set of needed BAPs to a reasonable size. Nonetheless, the impact of the BAPs with regard to their applicability for manipulation of 1D and 3D deformable objects



(a) contact established (b) displace upper page (c) touch upper page's edge (d) separate pages

(e) two pages grasped (f) displace (g) release lower page (h) upper page separated

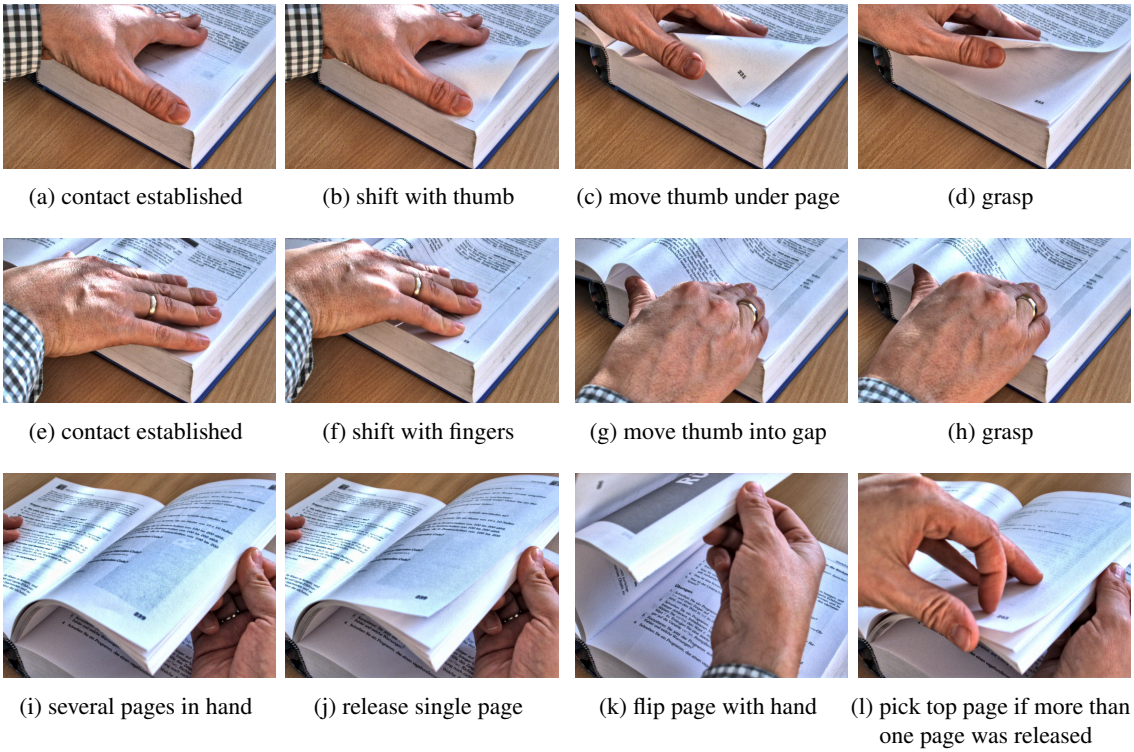
Figure 6.38: One-handed flicking by grasping the bottom right corner of the upper page. **(a-d)** After establishing contact, the top page is shifted towards the bottom of the book, so that the top-most page can be separated in order to grasp it. **(e-h)** Possible extension of the technique is shown in (a-d). Already grasped pages are held between the fore and middle fingers while the forefinger and thumb continue flicking. In cases in which several pages are accidentally grasped, a *PinchDisplace*-movement is used to separate the upper page.

is discussed later in Section 6.5.6. The following list represents the set of chosen manipulation examples for 2D deformable objects:

- Flicking though a book
- Putting a letter into an envelope
- Folding a piece of cloth
- Putting a slice of cheese on a slice of bread (including the opening of the packages)

Flicking Through a Book

Even though it could be argued that the ability to flick through the pages of a book is becoming more and more obsolete, especially for an agent that theoretically has access to the contents of every publication in existence through internet, this example was chosen as its underlying interaction defines a prototype for the separation of aligned layers of deformable planar objects. There are many other domains, such as when dealing with loose sheets of paper, food, banknotes or laundry, in which this ability is needed. Actually there are many types of flicking actions developed by people depending on the particular situation or goal, such as the type of the book, the thickness and the friction of the pages (anticipated and observed), whether the pages have to be handled carefully and whether we are performing a specific search (content-based or a page number) or aim to simply get an overview of the book's content. From the vast set of possible flicking techniques and individual variants of these, four common ones (the first one is presented in Figure 6.38 and three further ones are shown in Figure 6.39) were selected for the subsequent discussion with regard to performing them with a robot. In the optimal case a combination of our defined BAPs are used to specify them. Figure 6.38a-d shows a very natural flicking technique that is commonly used to traverse page by page into a book. The 3D structure of the book necessitates the generalization of the *Fixate* primitive to block the book's movement into a particular direction. With this, the displacement of the top-most page (see Figure 6.38b) can be achieved employing the *Swipe*-primitive. However, as our original definition of *Swipe* was designed to swipe *over* the



(a) contact established (b) shift with thumb (c) move thumb under page (d) grasp

(e) contact established (f) shift with fingers (g) move thumb into gap (h) grasp

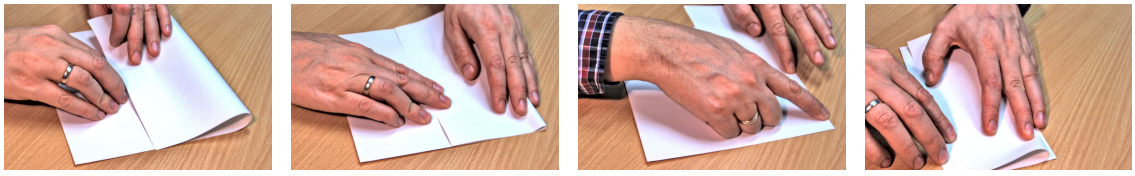
(i) several pages in hand (j) release single page (k) flip page with hand (l) pick top page if more than one page was released

Figure 6.39: More page flicking techniques. **(a-d)** The page is fixated with the fingers while the thumb is used to bulge up the top page. **(e-h)** Analogous to (a-d), but now the thumb fixates while the fingers are used to bulge up the top page (requires the other hand). **(i-k)** Faster flicking that offers a good overview of the page content but needs preparation to get to the situation depicted in (i). Single pages are released by a minimal rotation of the thumb's tip. **(l)** If more than one page is accidentally released, the free left hand can be used to pick off the top-most page.

paper, another extension is needed here. Similar to the *PinchDisplace*-primitive, the contact force must be adjusted to ensure that the paper is shifted along with the finger movement. To avoid confusion with swiping movements, we introduce this functionality as the *Shift*-primitive. *Shift* can seamlessly be blended into *Fixate* by stopping the movement and by increasing the contact pressure if needed. The separation of the top-most page requires very sensitive tactile feedback, whose contact-shape must be maintained while moving the thumb upwards (see Figure 6.38c). As this seems to be another common sub-step for paper-manipulation, it is used to define another new primitive, *EdgeGrasp*. The *Grasp* component of its name underlines the fact that it is mostly used to pinch-grasp the corner or the edge of the lifted page (see Figure 6.38d).

Figure 6.38e-h shows a commonly occuring error situation, in which more than one page is accidentally grasped using the previously defined flicking method. This situation can be resolved using an extended *PinchDisplace*-primitive that is able to release the displaced layer touched by the thumb by translocating the pinch-grasp center away from the thumb's tip (see Figure 6.38g,h). A remaining question is how a robot system would be able to detect, whether one or several pages were grasped. Assuming that the thickness of the grasped layers cannot be measured accurately enough to distinguish between one and two layers, a probing movement similar to *PinchDisplace* could be performed in order to detect a single page based on slip-detection as was presented by Schöpfer et al. [2010].

Figure 6.39 shows three additional page flicking techniques. The first two techniques (see Figure 6.39a-d and e-h) could be achieved using the new *Shift*-primitive, followed by *Grap* executed subject-to the *Fixate*-primitive parametrized with the end-posture of *Shift*. The third flicking tech-



(a) center-align corner (b) prepare fold (c) crease fold line (d) perform second fold

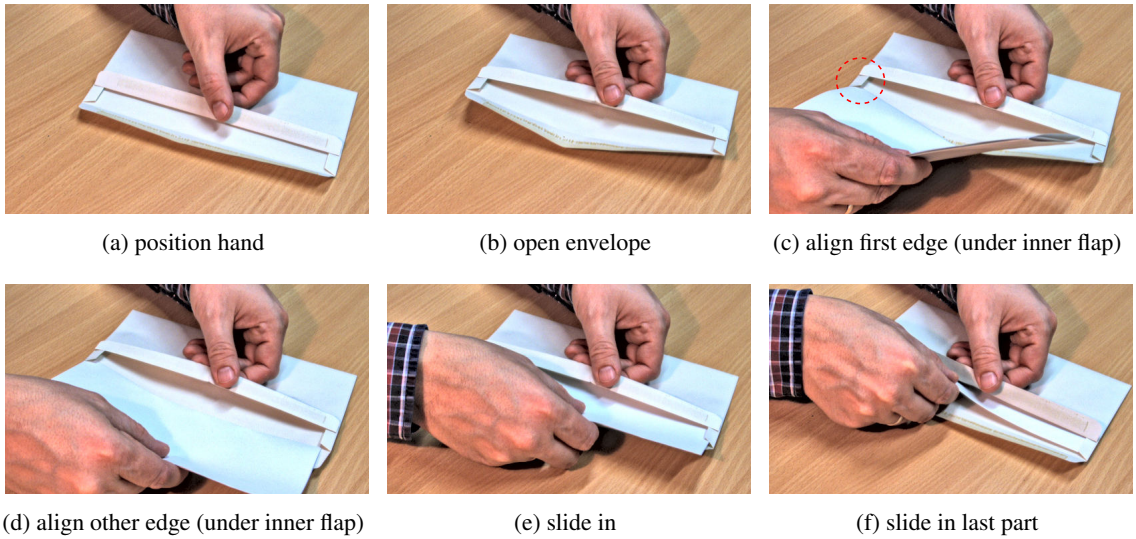
Figure 6.40: Folding a paper in thirds. **(a-c)** The main difference to the previously discussed sequence of folding paper in half is the more advanced alignment that is needed in step (a) and the second fold that has to be applied (d).

nique (see Figure 6.39i-l) requires an extra action to arrive at the initial situation depicted in Figure 6.39i. To this end, our first flicking technique (see Figure 6.38a-d) must be slightly adapted so that not only the first, but a whole stack of pages are lifted to allow grasping with the right hand. The most difficult step is depicted in Figure 6.39j. Here, a minimal rotation of the thumb-tip is performed so that only the top-most page is released. The extreme dexterity that is needed for this step is underlined by the fact, that it also often happens that we accidentally release more than one page at a time. In this case the robot could, in a similar fashion to a human, switch its operation mode to separating the released pages with the free hand (see Figure 6.39l). While this error handling could be implemented by re-using the *EdgeGrasp*-primitive, a very special *ReleaseTopPage*-primitive needs to be introduced to perform the releasing action.

Putting a Letter into an Envelope

While folding a sheet of paper in half was already solved both theoretically and even using our robot, folding into three equal parts, as is needed when fitting an A4 sheet of paper into a C 5/6 (DIN-lang) envelope requires an additional set of actions to be carried out, including an under-specified *Align*-step. For the initial fold, a paper corner must be aligned with the adjacent long edge so that the target point splits the original paper in equal thirds. Given a good modeling and detection module, a robot system is likely to solve this task even more precisely than humans, who easily end up with *thirds* that differ in the order of a centimeter in their sizes. While this is, given the additional height of the envelope[13], usually sufficient, a robot could easily reach millimeter accuracy here. Figure 6.40 shows how a human would solve this tasks. By extracting the desired alignment coordinates from the model, a robot could carry out the initial fold (see Figure 6.40a-c) like the initial paper aeroplane fold. The second fold (see Figure 6.40d), prepared by turning the paper over using the *RigidMove*-primitive, can be performed in a similar fashion. To this end, the already folded paper can be treated as an unfolded sheet of paper with an altered aspect ratio.
In contrast to this, the actual insertion of the folded sheet of paper into the envelope (see Figure 6.41) is indeed very different from the interactions that were considered for the initial definition of BAPs. Even though, an apparently very special one-handed interaction is needed to open the envelope (see Figure 6.41a,b), it can be modeled by a simple cascade of existing BAPs. To this end, *Grasp* is executed *subject-to Fixate* using the required pinch-grasp prototype with one hand. While maintaining the grasp, a *Move*-primitive can be used to open the envelop, leading to a final cascade of three primitives: $Move \lhd Grasp \lhd Fixate$. Once the envelope is opened (see Figure 6.41b), the previously folded paper can be inserted. After grasping the paper appropriately (see Figure 6.41c) using one of the already presented picking-up techniques, the insertion is initialized by aligning one short-edge with the corresponding inner edge of the envelope. The desired edge-

[13] A 297mm A4 page, folded in 99mm thirds, put into a 110mm envelope allows for an inaccuracy of about one centimeter.



(a) position hand (b) open envelope (c) align first edge (under inner flap)

(d) align other edge (under inner flap) (e) slide in (f) slide in last part

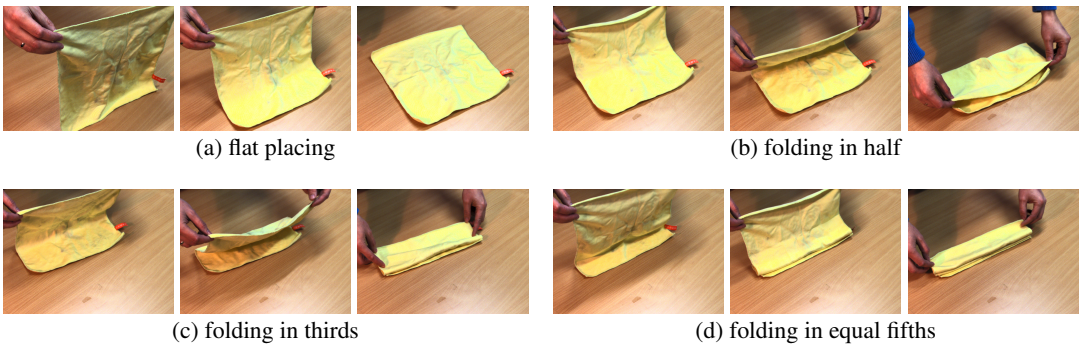
Figure 6.41: Inserting a folded sheet of paper into an envelope. **(a,b)** The envelope is opened with a special one-handed combination of grasping, lifting and fixating. **(c,d)** The insertion is performed in two steps, aligning first one and then the other edge of the paper. The paper must be brought *under* the envelope's inner flaps (see marked area in c). **(e,f)** For the last fraction of the sliding-in movement the hand that grasps the paper must be moved slightly upwards to avoid crumpling the bottom part of the envelope's closure flap.

to-edge alignment is particularly difficult as the paper corner must be brought under the envelope's inner flap (see marked area in Figure 6.41c). This is achieved by first aligning the paper edge vertically with the envelope, followed by aligning the edges, which can be expressed by cascading two instances of *Align*: *Align*(edges) $\lhd$ *Align*(vertical). The same action must be mirrored to insert the opposite paper edge, while ensuring to maintain the existing alignment: *Align*(right edges) $\lhd$ *Align*(right vertical) $\lhd$ *Align*(left edges) $\lhd$ *Align*(left vertical)[14]. Shifting the paper into the envelope is achieved using a rather simple *RigidMove* primitive. For the last 20% of the movement, the grasping hand must be slightly lifted to avoid crumpling the bottom part of the envelope's closure flap with the bottom fingers, which can, however, be trivially realized using a 3-step *RigidMove* trajectory (shift, lift, shift).

Folding a Piece of Cloth

Due to its high relevance for potential household robots, laundry folding has become a very popular sub-domain of robotic manipulation of deformable objects. The main difference to the manipulation of paper is the softer nature of cloth leading to a basically negligible bending stiffness and shape permanence. While gravity could be neglected almost totally in the case of paper, it becomes a fundamental factor for the manipulation of cloth.
Transitioning to a purely gravity based method leads to a total shift in the actual difficulties the robotic system needs to address. While we could reasonable expect in most situations to start out with perfectly flat sheets of paper, cloth is very prone to crumple, leading to the conceptional necessity to include straightening as an initial step of the manipulation sequence. However, as the straightening out itself is a very complex task, existing work has largely focused on untangling and straightening of laundry objects [Bersch et al., 2011; Maitin-Shepard et al., 2010], or simply assumes the to-be-folded laundry be straightened already [Miller et al., 2012; Van Den Berg et al.,

[14] *Left* and right are here used with respect to the images in figure 6.41



(a) flat placing (b) folding in half

(c) folding in thirds (d) folding in equal fifths

Figure 6.42: Visualization of the *GravityPlace*-primitive. The number of folds that are added depends on the horizontal movement that is performed while slowly lowering the hands.

2010].

The straightening of crumpled laundry objects is determined by a possibly proactive search for the corners for the object. In the following, we assume that the objects are crumpled only, so that more difficult situations, in which the object is knotted or even partly turned inside is ignored. Thus, once two adjacent corners are identified, the enclosed edge can be stretched so that the gravity automatically straightens the entire piece. In cases in which the edge is twisted, re-grasping might be necessary. The identification of the object corners is closely linked to the capabilities of the visual detection and modeling module that has to be able to bootstrap itself given an initially, perhaps strongly, deformed object. Under the concededly unrealistic assumption that an accurate model is given under these circumstances, the corners can either be grasped directly, or they have to be uncovered by picking and placing other parts of the object appropriately, which both can be implemented using our *Grasp* and *Move* primitives. However, as even for humans it is naturally not possible to extract the complete deformation of a crumpled piece of cloth, it must be assumed that the vision system can only provide a very coarse impression of the object deformation from an undisturbed object. This leads to the necessity to proactively move and deform the object to reveal obvious key-points such as texture, corners, collars or buttons which allows the initialization of an internal model as was shown by Maitin-Shepard et al. [2010]. However the actions that are needed for this cannot be classified to be actually *dexterous* as they can be fully described using sequences of *Grasp* and *Move* actions that could be performed just as well using a simple two-jaw-gripper.

Similar to the domain of paper folding, (gravity-based) folding of laundry consists of two almost orthogonal aspects, the planning of folding sequences and the actual generation of robot trajectories to execute the folding actions. Planning is conceptually very close to the planning of origami [Miller et al., 2012], which was extensively reviewed in Section 5.1.4.

For the generation of robot trajectories for laundry folding, inertia is often considered to be a negligible factor, which is, however, only true as long as the robot movements are carried out slowly. The faster the robot moves, the higher the influence of mass and inertia that has to be taken into account during the trajectory generation. Due to their intrinsic similarity, we combine gravity-based folding and gravity-based placing into a new *GravityPlace*-primitive (see Figure 6.42).

In order to accelerate the folding process, faster robot movements are required. Thus, the resulting *swinging* of the loose object ends caused by inertia must be taken into account for the trajectory generation. In an optimal case, the planning process should not only try to counteract these effects, e.g. by a controlled slowing down of movements, but should also try to exploit them to improve the folding speed. This, however, requires the system to have a good anticipation of the object's physical mass, mass distribution and stiffness – properties that can be extracted by weighing the object using force-torque sensors after lifting it. In addition, a sophisticated physical simulation is needed for the planning.

It is important to underline that folding is not the only possible interaction that can be imagined with laundry. In particular, with regard to the care of elderly people (e.g. the Care-o-Bot [Graf et al., 2004, 2009; Hans et al., 2002]), the ability to be able to dress people will also become important. When dealing with clothes, the topology of an item is very important, but such considerations lead to a large set of additional challenges for perception, modeling and the robot-control. The work of Lukas Twardon [Twardon and Ritter, 2015b], who enabled a robot to grasp and hang up a knit cap using a bi-manual anthropomorphic robot, is particularly relevant to this topic.

Putting a Slice of Cheese on a Slice of Bread

A completely different set of deformable objects are food items. For the chosen example of putting a slice of cheese on a slice of bread, the actual appropriate placing of the grasped slice of cheese can be trivially realized using the *GravityPlace*-primitive. However, given a package of sliced bread, packaged cheese and a plate, both bread and cheese must first be unpacked, which provides, due to the compliant nature of the packaging materials used, even further examples for the manipulation of common deformable objects. Therefore, in order to get a deeper and more complete understanding of the needed preparation-steps, the opening and the closing of the packaging is added to the following discussion. Figure 6.43 shows the course and the sub-steps of this task. To remove a slice of bread, the packing's clip must be opened, a slice of bread must be picked off from the stack and ideally the packaging should be sealed again by closing it with the previously removed closure clip. A similarly difficult action sequence is needed to pick a slice of cheese from an initially sealed packet of cheese.
In contrast to the other examples that have been discussed so far, we here deal with numerous objects of different types. In order to focus on the robot control aspects, we assume that the visual detection and modeling system has been generalized appropriately so that the objects can be detected and their deformation can be tracked during the course of the manipulation sequence. However, some particularly demanding visual detection tasks are addressed in order to call attention to the most obvious difficulties for this domain.

The first sub-task is to open the closure clip that seals the bread packaging (see Figure 6.43a). While for the visual detection of the packaging, reflections and transparency must be handled, the detection of the closure clip is difficult because of its small size – in particular given the large amount of occlusion that has to be dealt with. As a preparation, the closed packaging must be oriented appropriately (using the *RigidMove*-primitive) so that the open end of the closure clip can be conveniently reached. While a very accurate visual detection could possibly allow the open end to be grasped using the standard *Grasp*-primitive, occlusion and the small size of the graspable end suggests a more sophisticated grasping technique that based on tactile servoing [Li et al., 2015] is needed. By generalizing *Grasp* to optionally maintain a desired tactile feedback through an altering of the grasp posture, the fixation of the clip could be maintained and optimized while *Move* is executed *subject-to* the *Grasp* primitive to unwind the closure clip. Once the closure clip is unwound, it can be removed by means of trivial picking and placing action.
The next sub-task is to open the packaging to be able to pick off the top slice of bread from within. The visual detection of the packing's opening could be simplified and enhanced using a dedicated topology-based boundary detection mechanism such as was implemented by Twardon and Ritter [2015a]. For the sake of brevity, the main part of the packaging, which contains the bread, will in the following be referred to as *body* and the open end that forms the opening will be referred to as *neck*. Due to the extreme deformation of the material around the *neck*, a heuristical initialization on the basis of the *Spread*-primitive (see Figure 6.43b1,2) might be necessary or at least very helpful. Picking off the top-most slice of bread from within the packing's *body* is conceptually very similar to the *EdgeGrasp*-primitive that was introduced for the page flicking example earlier in

Figure 6.43: Putting a slice of cheese on a slice of bread includes opening and closing of food packages. The complete interaction demands the robot to be capable to deal with several most different types of deformable materials: bread, cheese, the wire in the closure clip and the different types of plastic found in the packages. A very important situation is depicted in (d4) in which an error situation must be detected an handled.

this section. The main difference here is that the severe space limitations might require the robot to separate the top slice using its finger-tips (see Figure 6.43b4), before pinch-grasping and pulling out the slice. While the bending stiffness of bread is close to the stiffness of paper, it introduces a very important property that has not yet been taken into account for the physical modeling framework: *tear strength*. Not only for physics-based planning, but also for closed-loop feedback-based controllers, the possible tearing of objects must be considered – most of the time in order to avoid tearing. In the present example the slice of bread must be extracted without damaging it. To this end, we generalize the primitives *Move* and *RigidMove* to optionally support tearing avoidance. Without claiming to provide an actual solution for such a complex functionality, it can be stated that a possible implementation would have to use all different kinds of visual, haptic and proprioceptive feedback available.

Subsequently, the packaging must be sealed again by wrapping the closure clip around its *neck*. This can be significantly facilitated by an initial twisting of the *neck* relative to the *body*, which leads to a narrowing of the *neck*'s diameter. A trivial implementation based on *Grasp* and *Move* would suffice here (see Figure 6.43c1). The wrapping of the closure clip around the now twisted *neck* (see Figure 6.43c2-4) is particularly difficult as it requires not only strength and dexterity at the same time, but also a very tight coordination of both hands. While the left hand[15] fixates the twisted *neck* to avoid any unwanted untwisting, it must also prevent the *body* from rotating, which can be achieved by gently pressing it against the table-top: *Fixate*(body) $\lhd$ *Fixate*(neck).

Now, the right hand must grasp the closure clip appropriately. The fact that the clip does not have to be completely straightened when removing it, its bent part can now directly be shifted over the packing's *neck* (see Figure 6.43c2). Otherwise, it would have been necessary to prepare the clip with two hands before *occupying* the left hand with the fixation. An appropriate grasp orientation can, if necessary, be achieved in a one-handed fashion by successive pick, place and re-pick actions. Once the closure clip's opening is shifted over the *neck*, the right hand's *Grasp*/*Move* (clip) merges into an extra fixation primitive (*Fixate*), which is possible by pinching the neck together using the clip (see Figure 6.43c3). Subsequently, the left hand uses the forefinger to bend the closure clip's open end firmly around the neck (see Figure 6.43c4). By introducing a new *Bend*-primitive, which realizes a possibly closed-loop controlled bending of an object, the fixation of the closure clip can be implemented using *Bend* $\lhd$ *Fixate*(body) $\lhd$ *Fixate*(neck).

In the next interaction step, the packaging of the cheese must be opened. This seems initially trivial (see Figure 6.43d1,2) as the opening flap can be easily picked using the *EdgeGrasp*-primitive. However, the high force that is needed to open the packing's top lid is likely to accidentally tear it (see Figure 6.43d4). While the previously introduced tear-avoidance-enabled *Move*-primitive would theoretically prevent the material from tearing, it remains unclear, how the system could be enabled to automatically alter the *Grasp* position and the *Move* direction to still fully open the packing's top lid. Such a situation could conceptually be approached either by employing physical simulation or by local optimization, assuming there is a small sub-space in which the system can probe the local tearing tension of the material by altering the *Grasp* and *Move*-parameters. Alternatively, the system would have to be endowed with an error handling mechanism, so that it could automatically infer task-dependent alternative *Grasp*/*Move* combinations to open the lid with minimal damage. However, the highly complex planning needed for this would also necessitate a well-parametrized physical simulation.

In order to pick off the top-most slice of cheese, once again *EdgeGrasp* is employed (see Figure 6.43f1,2). The lifting of the slice (see Figure 6.43f3) using *Move* with tear-avoidance enabled underlines that even friction forces and the stickiness of the lifted food item with respect to its tear-strength can be significant factors in avoiding tearing. The final appropriate placing of the cheese on the slice of bread (see Figure 6.43f4,5) can be performed with *GravityPlace* and the closing of the cheese-packaging can be achieved using *Move* to roughly fold back the packing's lid (see

[15] *Left* and *right* here refer to the examples depicted in Figure 6.43 – they could be trivially swapped

Figure 6.43g1-3), followed by *Swipe* to ensure the lid remains closed (see Figure 6.43g4,5).

6.5.5 Discussion

We presented an idea of a generic robotic system for the anthropomorphic manipulation of planar deformable objects. The system is based on an extendable set of *basic action primitives* (BAPs), each of which allowed a certain parametrized action to be performed. For some interaction steps, BAPs had to be sequenced, executed in parallel, or put into a precedence cascade using the *subject to* relation. Starting with an initial set of BAPs that were extracted from a paper aeroplane folding sequence, several other manipulation examples were *run through* in order to find out whether the existing primitives sufficed for the realization of these examples. Some primitives had to be extended or generalized and sometimes new primitives were required. However, a convergence effect could be detected over time. That is, the more examples that were taken into account, the less likely adaptions to the set of BAPs were required.
By design, we here concentrated on the definition of BAPs and their potential combination, which even allowed us to conceptually enable the robot to open food packaging and to handle food items. A next possible step would be to develop an automatic BAP sequencing mechanism that combines and parametrizes the existing primitives to achieve a specified target object configuration. To this end, Probabilistic Roadmap (PRM)-Methods combined with a physical simulation [Bayazit et al., 2002] or simulation-based planning approaches [Yoshida et al., 2015] could be employed. However, as the bad predictability of the actual outcome of manipulation actions on deformable objects leads to an almost non-deterministic deformation behavior, it is likely that not only some of the primitives themselves, but also the planning and sequencing of actions will have to be implemented in a closed loop manner. The research towards finding appropriate BAPs to realize the selected action examples also provided valuable insights into the inherent difficulties that have to be dealt with when manipulating planar deformable objects. Furthermore, in addition to the most obviously common and rather generic primitives *Grasp*, *Move*, *RigidMove* and *Fixate*, the desire to re-use existing primitives revealed other re-occuring actions, such as the ones described by the *EdgeGrasp* or the *GravityPlace* primitives.
The manual extraction and definition of BAPs as basic constituents of complex interactions is not only a difficult and time consuming task, but due to the lack of well defined rules that tell us how to segment a perceived interaction into primitives it also has a rather random or arbitrary characteristic property. As an alternative, many research projects focus on the automatic extraction of action primitives from demonstrated actions. This led Schaal [2006] to introduce *Dynamic Movement Primitives* (DMPs) – a framework that represents given input trajectories on the basis of the dynamics of differential equations that allow start and end configurations of the trajectories to be altered. Many later research projects took on the idea of DMPs and developed a large set of extensions, such as for the automatic learning of primitive functions and parameters [Ijspeert et al., 2013; Pastor et al., 2009, 2013], the coordination of different robots [Zhou et al., 2016] and the planning of collaborative tasks between robots and humans [Cui et al., 2016; Maeda et al., 2014, 2016].
However, the close coupling between perception and robot actions in many of our BAPs poses the question of how to integrate highly complex visuo-haptic sensor feedback into a DMP-framework in an online fashion. This could necessitate a hybrid approach, in which single BAPs are internally realized on the basis of DMPs. While progress is being made, a long road lies ahead until robots possess the dexterous capabilities we would like them to have.

6.5.6 Generalization to 1D and 3D Deformable Objects

By initially arguing (see Chapter 1) that the field of the manipulation of generic deformable objects is too versatile, we decided to narrow the domain by restricting the manipulated type of objects. Thus, we first assumed objects to be planar and we put our main focus on the manipulation of paper. With the development of a concept for a generic system for the manipulation of paper and paper-like objects (see Section 6.5), we showed, that – conceptually – a small set of *basic action primitives* (BAPs) suffices for the realization of very complex manipulation actions. By employing the primitives, which were originally *derived from* and therefore also *tuned for* paper manipulation, to the handling of other kinds of deformable 2D objects including plastic packaging material and even food items such as cheese, their generalizability was investigated (see Section 6.5.4).
An obvious next generalization step is to apply our set of BAPs to the manipulation of 1D and 3D deformable objects. Some of the most common types of 1D deformable objects that we deal with on an every day basis are cables and threads (including thin threads, common USB, headphone and power-cables and thick ropes). As discussed in Section 1.1.2, the robotic handling of 1D deformable objects commonly requires a specialized 1D model. The adaption of the presented Kinect-based detection mechanism for planar objects (see Section 6.2) combined with an appropriate physics-based linear deformable model, seems straight forward. However, the presented SURF-feature-based extension that provides more reliable associations between real-world points and object coordinates would have to be replaced and for thinner objects an alternative 3D sensor with a higher spatial resolution would be needed. The quite similar system presented in Schulman et al. [2013b] used a thick rope and required different patches of that rope to be tinted in well distinguishable homogeneous colors. Assuming a reliable detection and modeling mechanism to be given, we can concentrate on the robotic manipulation. As an example, the untangling of a cable is considered. Similar to the folding domain (see Section 1.1.3), it seams reasonable to split the problem into the three sub-problems: theoretical solving of the required untangling, generation of the required model deformation trajectory and the generation of robot movements that accomplish this. However, the complete theoretical solving of the untangling strongly assumes a reliable guess of the initial model deformation, which might be very difficult to extract from a fully entangled cable. Therefore, an iterative untangling strategy, similar to how a human would perform this task, might be needed. Once such a strategy is available, the manipulation itself could most likely be performed using the *Grasp*-primitive, which is appropriately set up with a desired tactile pattern to ensure a stable grasp of the thin cable, and the *Move*-primitive alternately applied by each hand. As another example for the handling of a linear deformable object, one of the most precise manipulations that humans are able to perform is the threading of a needle, which remains, due to the high accuracy needed, a difficult challenge for robots. In 2015, Huang et al. [2015] provided a setup that is able to thread a needle with a thread of a diameter of 1 to 4mm, but they introduced a custom designed robot that spins the thread fast enough to fully straighten it, before the actual insertion is performed. Even though industrial robots with sub-millimeter accuracy exist, the threading of a needle seems to be a typical example requiring a closed-loop visual-servo controller, which, in this case, must be combined with an appropriate macro lens to enable the robot to visually guide the task. However, from the perspective of dexterous manipulation, the task can once again be trivially described using our *Grasp* and *Move* primitives.
The tying of a bow could be deemed as example that particularly requires not only accuracy, but also a high level of dexterity. A possible planning approach that uses two *needle-like* grippers, was proposed in 2006 by Saha and Isto [2006] and there is even a patent for a custom designed *bow making apparatus* [Monahan, 1993]. However, the task was thus far not solved for an anthropomorphic robot hand, which would, given the ease and speed with which humans can tie a bow, be a very desirable skill for a robot to handle. Once learned, human bow tying seems to be mostly reflex-driven rather than an action that requires intense action planning. Furthermore, visual feed-

back is likely to play only a secondary role as the action can easily be performed without looking but trying to tie a bow while wearing thick gloves, which severely reduces our tactile sensibility, is extremely difficult. Even neglecting the fact that an anthropomorphic robotic solution would have to deal with the – in comparison to the human hand – typically strongly limited touch sensitive area, an easy implementation on the basis of our BAPs seems not possible as the whole bi-manual manipulation sequence would most likely have to be modeled as one complex tactile-servoing-controlled action.

Typical examples for commonly handled 3D deformable objects are pillows, mattresses, sponges, loaves of bread, soft toys and inflatable items such as balloons or swimming pools. However, in contrast to their 1D or 2D counter-parts, typical manipulations with 3D deformable objects are neither distinguished by a high accuracy nor by high dexterity. Instead, they are often either only grasped, placed and fixated or they are manipulated with rather coarse interactions. There are two main reasons why humans usually do not use 3D deformable objects in a highly precise manner. First, the objects' compliant nature allows actions to be performed with less precision and second, the deformability of the objects impairs our prediction accuracy so that we might not be able to perform the task in the first place.

Another class of deformable objects that has not yet been taken into account is defined by prototypes such as dough or clay. Objects that consist of such materials are not only continuously deformable, but they also display plastic behavior. In addition, parts can be removed and later be attached somewhere else. Going even one step further, fluids could be classified as 3D deformable objects that are characterized by different properties, such as surface tension and viscosity. However, even though these examples reveal many interesting research questions for the field of anthropomorphic robotic manipulation, they will not be discussed in more depth here as the presented concepts for detection, modeling and robot control cannot trivially be ported.

Before we conclude, a very important, but perhaps not obvious type of 3D deformable *object* that humans deal with every day must be examined: *other humans*. In addition to surgical robots, in particular with regard to geriatric care, possible assistance robots have to be endowed with the ability to interact with human bodies. The required level of detail of an employed model here heavily depends on the actual application, but the use of a soft-body physics-based model will no doubt be required. While our set of BAPs would require a number of additional aspects such as *caution* and *comfort*, our vision system could be extended to track people without major adaptions.

7 Conclusion

Having presented initial results for general extensions of our robot paper manipulation system (see Chapter 6), we now provide a short wrap-up and discussion of our contributions (see Section 7.1). We conclude with some final thoughts and finish up by providing a summary of possible future directions (see Section 7.2).

7.1 Summary & Discussion

By iteratively extending our initial robot system for shifting paper (see Section 2.2) along *Perception*, *Modeling* and *Robot Control* axes, we developed several systems for dexterous anthropomorphic manipulation of paper and paper-like objects. The realization of these systems required us to tackle numerous previously unsolved challenges, such as the tracking of paper while it is being manipulated by humans or robot hands, the real-time modeling of paper deformation including the update of the model parameters on the basis of perceptual input and the development of robot systems that can execute complex paper manipulation actions. During this process, the requirements for the *next* system and the knowledge and the insights gained from the *current* system mutually supplemented each other, and allowed us to conceptualize and implement increasingly complex systems. *Perception*, *Modeling* and *Robot Control* aspects were always closely coupled so that in each iteration of the system, the complexity of the required world and object models and the corresponding detection systems were driven by the aspects of the paper that the robotics application was about to change.

The Image Component Library (ICL)

During the different iterations of the robotic system, our computer vision library, ICL (see Chapter 3), was always a crucial prerequisite for the realization of ideas and prototypes developed for the perception and modeling domains. ICL allowed us to implement all applications presented in this thesis from the ground up. The integration of the developed algorithms and software components into ICL, not only ensured their re-usability, but also demanded that we severely reconsider and optimize the generality of the existing and new classes and interfaces.
Through its unmatched combination of speed, ease of use and function volume, ICL was shown to have a leading position among existing computer-vision libraries and we note that beyond the work presented here, it has been employed in many projects in and outside Bielefeld University. Its main unique characteristic is its widely spread function volume that not only covers computer-vision related functions and classes, but also a large set of tools that facilitate the development of complex interactive and real-time capable applications. This allows diverse applications to be implemented *in ICL* using a single, consistent, API.

Shifting Paper

Our initial system for shifting paper (see Section 2.2), treated the paper as a rigid object and altered the model position and orientation in 2D space only. Thus, a 2D model described by a 3D feature vector[1], combined with a fixed paper size, sufficed to fully specify the relationship between detected paper corners and edges and their associated counterpart in the model. The restriction to 2D and the assumption of a top-view allowed the detection to be simply performed in 2D image space. Using a simple color-edge detector, the corners and the edges of the paper could be sufficiently tracked even in the presence of occlusions. The limited reachability of our robot setup led to a comprehensive reachability-study that helped us to define a shifting posture as a trade-off between optimal finger placement and maneuverability. The implementation of the contact establishment for the shifting action revealed that the contact force measured by the existing touch sensors heavily depends on the alignment of the sensors with the contact surface and thus is not the ideal feedback channel to achieve and maintain a desired contact force. Even though the compliant nature of the Shadow-Robot hands meant higher inaccuracy as the paper was released, the final system was able to satisfactorily align a sheet of paper with a desired position and orientation on a 2D worktop employing several online-planned perceive/action steps.

Picking up Paper

Our next iteration was to enable the robot to bi-manually pick up paper (see Chapter 4). The picking up action was realized by bulging up the paper with one hand so that the other hand could pinch-grasp the bulged-up center. While our previous 2D detection module would have been sufficient for the detection of the paper's initial pose, the robotic grasping of the bulged-up edge required 3D information. Not only with regard to further manipulations that were planned but also in order to optimize the generalizability of the detection and modeling mechanism, a deformable 3D paper model was preferred to a heuristical solution that only estimates the xyz-position of the paper's bulge. Given the severe amount of occlusion that was expected, multi-camera registration and 2D/3D-keypoint detection and association issues were bypassed using a newly introduced type of fiducial marker that were printed directly on the paper. Using a calibrated multi-camera setup, the resulting correspondences between 2D paper space coordinates and 3D world coordinates were used to compare a mathematical model with a physics-based model. A detailed comparison, conducted using a realistic 3D simulation, of the performances of these models in different paper manipulation scenarios showed that the simpler mathematical model performed as well as the physics model in many situations and even showed better accuracy in some cases. However, due to the physical model's superior properties when handling occlusions and its extensible characteristics, the detection system for the robotic picking-up of paper experiment employed the physical model.
The development of the robot control system underlined the fact that a mechanism for the native integration of closed-loop controllers was missing in our existing system. By replacing the touch-sensor based contact establishment mechanism, employed in the shifting system, by measuring finger joint-angle discrepancies with respect to a base posture, the risk of damaging the hand due to too high a contact force being produced was significantly decreased. However, as the associated closed-loop system was implemented indirectly through out the network-based robot control interface, the robot's movements were slow and suffered from excessive jerk. The slightly disappointing overall success rate of only 60% was mainly caused by the internal tension in the pneumatic muscles of the Shadow robot hand that often caused a dislodging of the paper from the pinch-grasp when the hand responsible for bulging up the paper released contact.

[1] x/y-position and in-plane rotation

Folding Paper

The next task we devoted ourselves to was to endow our anthropomorphic robot with the ability to fold paper (see Chapter 5). This required us to tackle several drawbacks of the previous system and to implement a set of additional features. Due to the limited detection accuracy of our custom fiducial marker layout, a first enhancement was to implement and include a more powerful marker design. A generic marker detection framework was implemented in ICL that includes plugins for all common fiducial marker types and we found that BCH-code markers provided the most optimal performance in terms of speed, accuracy and false positive rates. Providing the previous physics-based model with the ability to be able to simulate folds by altering model constraints along fold lines, coupled with a generalized model control law, allowed us to significantly improve the capabilities of our detection and modeling framework. After extensively evaluating the performance of the framework by tracking several human paper manipulation sequences, a set of extensions required for the robot control framework were identified.
By extending the arm-server so that rigid objects can be registered and their pose can be updated at run-time, robot movements can be specified relative to an object frame. Along with a parametrized embedding mechanism for sub-HSMs, these extensions provided a significant simplification of the required HSM description code. In particular, this embedding mechanism was employed for the C_{touch} controller for joint angle feedback based contact establishment, which allowed us to significantly reduce the implementation overhead arising form the former explicitly coded joint-angle discrepancy based perceive/action loops. In addition, the C_{force}-controller based *active postures*, which maintain a desired touch-sensor contact force by altering the finger flexion angles, were added in order to facilitate fixating, swiping and shifting. These extensions facilitated the development of the anthropomorphic paper folding system. By assuming the initial paper rotation is close enough to the desired orientation for the start of the folding action[2], the initial positioning can be reached using of the new contact establishment and force-maintaining controllers in combination with a single joint shifting and rotation movement. Once positioned correctly, the paper can be registered in the server so that all following commands can be executed relative to the paper coordinate frame, which was continuously updated by the visual input. By these means the system implicitly adapts to all unintended paper movements. While the implementation of the actual folding action using a predefined and hand-tuned movement trajectory allowed an acceptable level of accuracy to be attained, a closed-loop mechanism that strives for an optimal alignment of the two folded paper corners and edges would be a desirable extension. The creasing was implemented using two successive swipe-movements, one that is carried out with four fingers utilizing a low active posture force, and a second one carried out with two fingers employing a high force to harden the crease.
Given the many steps of the manipulation sequence, the system's final overall success rate of 80% was very satisfactory.

Advanced Aspects

The iterations of the system for shifting, picking up and folding paper provided valuable insights into the field of anthropomorphic robot paper manipulation. In order to ground the discussion about the opportunities and the limitations of some next steps, a set of prototype-systems were conceptualized and developed (see Chapter 6). As a first step, the physical paper model was extended so that arbitrary folds can be represented accurately and the model control law was further generalized to be able to handle the resulting irregular grid of nodes.
While our initial marker-based detection systems showed a very good tracking accuracy, their

[2] Otherwise, several reaching back and forward rotation actions are required (see Section 2.2)

real-world applicability remained questionable. Even though certain industrial or research applications might be able to cope well with the necessity to endow manipulated objects with markers, the approach must be seen as an intermediate step that eventually must be replaced by a markerless system if the aim is to employ such systems in general environments. The expensive and precisely calibrated camera-setup that is required, along with the resulting large footprint of the to-be-processed data poses an extra drawback that not only leads to high hardware costs, but also necessitates a large hardware setup. Our advanced system that employs a single Kinect device for the tracking (see Section 6.2) uses an ICP-variant on the RGB-D point cloud. Initial results were promising, but the lack of reliable associations between point cloud points and 2D points in the paper-space yielded unacceptable results in many situations. By extending the system to additionally employ 2D SURF-features that are mapped into the point cloud (see Section 6.3), a remarkable improvement in the tracking performance was achieved – even when realistically printed paper textures were used.
A remaining drawback of the visual detection system was the fact that fold lines had to be manually added to the model. An extensive study of common and custom designed curvature metrics revealed that automatic inference of when a fold is added to the model is very difficult to achieve. However, by assuming that a possible temporal fold onset is known by the system, a developed particle-based prototype, in which each particle represents a fold/non-fold hypothesis, was shown to not only provide good estimates of a fold's geometry, but could also be used to figure out when the paper had not been folded.
Finally, a concept of a generic anthropomorphic robotic manipulation framework for paper and paper-like objects, based on an extendable set of hand-crafted *basic action primitives* (BAPs) was developed. An initial set of BAPs was iteratively extended and generalized in order to realize increasingly complex manipulations by combining the BAPs successively, in parallel and in a hierarchically cascaded fashion. By the end, a comprehensive discussion revealed a promising generalizability of the BAPs with regard to the manipulations of some more general, non paper, 1D and 3D deformable objects. However, it also underlined the fact that many special manipulations such as the tying of a bow, require additional highly specialized BAPs that heavily rely on a closed-loop visuo-haptic feedback being available.

Final Thoughts

In this thesis, the challenges and opportunities for dexterous robotic manipulation systems for paper and paper-like objects were investigated. The results provide a snapshot of what is currently possible, and includes insights into what is being actively researched and indeed what advances we can expect to see over the coming years. Using our novel Computer-Vision library, ICL, a very powerful and extendable detection and modeling system was provided that has the potential to become a basis for further projects in this domain. The system cannot only be employed as a source of reliable online-feedback for robotic manipulation systems, but it can also be used to enrich studies about human manipulation of paper and paper-like objects. The developed robotic systems reveal, how well manually tuned heuristic interaction primitives, optionally implemented employing closed loop feedback, can be combined to realize various capabilities for dexterous anthropomorphic manipulation of paper. By conceptually extending this bottom-up approach of combining basic action primitives to achieve increasingly complex and generic interaction skills, we were able to show that such a system is indeed promising to be used to perform even the most difficult dexterous manipulations.
The research described in this thesis provides a significant contribution to the field of dexterous anthropomorphic robotic manipulation of paper and other deformable objects and thus takes us a couple of steps closer to the vision of early roboticists who dreamed of having general purpose robots in every home.

7.2 Outlook & Future Work

Although the extended systems (see Chapter 6) provide significant improvements that in places reach far beyond the capabilities of the final fully-fledged robotic system for paper folding, there are still countless possible routes towards further expansions of capabilities and improvements.
Our final physical paper model (see Section 6.1), which is able to precisely represent folds, constitutes a promising start for a large set of further possible enhancements. By adding further modeled properties of actual paper and other 2D deformable objects, such as tearing and cutting, surface friction, stretching (e.g. for plastic object) or the effects of moisture absorption, even more realistic behavior could be reflected and thus allow additional manipulations to be tracked. Another possible extension could more explicitly deal with folds. Given the numerical instabilities arising from several layers of paper that are stacked through folding, a mechanism that automatically aggregates these layers into a simpler (if necessary *thicker*) polygon, would not only allow for a significant reduction of the computational power needed, but also increase the system's overall stability. However, we note that in situations in which it is not known in advance whether a fold will be undone at a later point in time, the actual geometry would have to be stored. In terms of generalizability, future work could also strive for a more generic model that can not only represent 1D, 2D and 3D deformable objects, but that would implicitly also allow parts of the object or even the whole object to be rigid. While the employed Bullet physics engine allows all these extensions to be implemented, the challenge in this regard is to develop a framework that allows such models to be easily created and refined – optimally even from visual input alone or facilitated by a robotic system that pro-actively explores an unknown object to deduce its configuration and physical properties. Such an approach could be even extended by developing an Internet database of models that can be queried to skip or at least bootstrap the model creation.
A first step to further enhance the Kinect-based tracking system would be to generalize its feature-detection mechanism. Even though the employed SURF-features allowed us to achieve a very good tracking performance, other feature types might still help to further improve the results. In particular given the small output of SURF-features for common paper textures (see Section 6.3.3), a plugin mechanism for other feature types that would even allow several different features to be used simultaneously would indeed allow for even better tracking accuracy. In addition to incorporating further standard 2D image feature types, such as ORB [Rublee et al., 2011] or BRIEF [Calonder et al., 2010], an explicit handling of color and object-edge/corner features also seems desirable. Furthermore, 3D point cloud features, such as FPFH [Rusu et al., 2009] or SHOT [Salti et al., 2014], could be included and do not require a computationally expensive mapping between the 2D image space and point cloud. An optimal system could be able to access a large set of such feature types in order to automatically select a minimal set with respect to different preferred system properties, such as optimal speed, optimal accuracy or low memory/CPU/GPU usage. If possible, the set of employed features types could even be adaptable at run-time, so that the system could for example automatically adapt to changing lighting conditions. In the context of the robotic system, a further improvement could be achieved by more explicitly taking the manipulating hands into account. Using the robot's forward kinematics, the positions of the fingers of the hand are known and therefore the point cloud segments that belong to the hand can be filtered out from the point-cloud. Even though this does not restore the parts of the object that were occluded by the hands, it would provide a significant improvement for the overall object segmentation. In addition, by explicitly adding an articulated object of the robot hands into the physical simulation, the hand-surface would implicitly interact with the object model providing an extra source for physically-plausible model updates. If sufficiently sensitive touch sensors are available, tactile information could be incorporated into the model to further enrich the feedback.
By bundling all different kinds of visual, proprioceptive and tactile feedback into a single world model that is set up and enhanced with many kinds of available a priori information (such as the

robot geometry and kinematics or world knowledge about the manipulated objects), the spatio-temporal integration performed by an overarching physics engine is likely to allow for the creation of a very powerful global inference mechanism. Such a system could prove invaluable for bootstrapping the development of future general purpose robotic systems.

A possible future path towards a system similar to the one sketched in Section 6.5 would be to actually start to implement the suggested basic action primitives. It must be admitted that the process of the actually implementing these primitives would most likely reveal insights that suggest a different segmentation into primitives of the overall required interaction skills. Moreover, reducing the primitives to an actual implementation level and the associated pursuit of encapsulating any re-occurring functionalities and *behaviors* is likely to yield an additional set of *underlying*, perhaps even more basic, or atomic low-level primitives. However, similar to the described process of combining existing primitives using a set of operators such as sequential or parallel execution or the execution of one primitive *subject-to* another one, this would not break the general bottom-up idea of this approach. The time-consuming nature of the process of the implementation of primitives, could be counteracted by creating an open-source data-base for such primitives that allows researchers from all over the world to contribute their own primitives or extend, employ or test existing ones. However, in order to simplify the exchange and the re-usability of the created primitives, a commonly accepted and more widely distributed robot control platform, such as ROS [ros] should first be agreed upon[3]. A predominant issue for the creation of such primitives is the fact that many labs work with different kinds of robot hardware, which includes different robot arms, hands, relative mounts of bi-manual systems and also a large variety of different sensor endowments. To still be able to transfer primitives between different setups, the primitives either have to be implemented in a generalized fashion, or – more realistically – for some primitives, different robot/hand dependent versions must be provided. Actually, given the fact that the diversity of robot hardware might partly lead to completely different implementations of certain primitives, it seems mandatory to regard a primitive more in the sense of a description of what it does rather than in terms of how it is actually implemented, which, in turn, increases the impact of our provided elaborations. In a final bottom-up hierarchy of interaction primitives, the bottom layers that strongly depend on the hardware can be viewed as an abstraction layer from the employed robot hardware. Higher-level primitives that are mainly defined by a combination of other primitives must not necessarily have to deal with these issues.

Once a set of primitives is available, automatically sequencing them to achieve a desired object state and the learning of such sequences or even the learning of parameters of a sequencing mechanism would come into focus. In a similar way to how humans learn, it might be necessary to combine different learning paradigms such as imitation learning, reinforcement learning and even autonomous learning from proactive exploration[4].

Stepping back from our proposed bottom-up system, the problem of dexterous anthropomorphic manipulation of deformable objects could perhaps be approached from a more holistic perspective. Through the recent advancements in neural networks [LeCun et al., 2015], deep learning methods have become very powerful and deep convolutional neural networks have been shown to approach human performance in many recognition tasks. However, it must be admitted that in contrast to pure recognition tasks, which achieve astonishing performances when trained in a supervised fashion with millions of training examples, the intrinsic combinatorial complexity of *interaction* in general is still far beyond what neural networks are capable of tackling at present. Even though it seems theoretically possible to create a neural network that, similar to the human brain, automatically combines supervised and unsupervised learning techniques in order to fully autonomously bootstrap a dexterous manipulation system, it remains unclear whether our current approaches for hard and software yield enough resources to actually achieve this.

[3] Our robot control framework is also being gradually updated to employ ROS.

[4] Note that we do not claim these to be fully disjunctive.

The enthusiasm to reduce the natural complexity between raw input sensor and output actuator signals to more realistically tractable dimensions, once again suggests a pre-structuring on the basis of well known constituents of interaction. Thus, a set of well developed primitive actions could even be employed together with techniques from the fields of machine learning, AI and neural networks.
Although there are many possible ways in which this research can be built upon to improve the abilities of anthropomorphic robot hands further and to even provide them with human-like and beyond manual intelligence capabilities, we can, with some degrees of confidence, state that the first significant steps of this endeavor have been taken with this work.

Bibliography

Bullet Library. URL `http://www.bulletphysics.org`.

ROS - The Robot Operating System. URL `http://www.ros.org`.

Roger C Alperin. A mathematical theory of origami constructions and numbers. *New York J. Math*, 6(119):133, 2000.

Elliot Anshelevich, Scott Owens, Florent Lamiraux, and Lydia E Kavraki. Deformable volumes in path planning applications. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 3, pages 2290–2295. IEEE, 2000.

D Arnold. Questions on shell theory. In *Proceedings of the Workshop on Elastic Shells: Modeling, Analysis, and Computation*, 2000.

Bradley Atcheson, Felix Heide, and Wolfgang Heidrich. Caltag: High precision fiducial markers for camera calibration. In *VMV'10*, pages 41–48, 2010.

Steven Ross Atkinson, Steven Ross Atkinson, Jin Song Dong, Alena Griffiths, Andrew Hussey, Ian Maccoll, Anthony Macdonald, Craig Mann, Ian Peake, and Gordon Rose. Formal engineering of software library systems, 1997.

Devin Balkcom. *Robotic origami folding*. PhD thesis, Citeseer, 2004.

Devin J Balkcom and Matthew T Mason. Robotic origami folding. *The International Journal of Robotics Research*, 27(5):613–627, 2008.

Bauckhage, Wachsmuth, Hanheide, Wrede, Sagerer, Heidemann, and Ritter. The visual active memory perspective on integrated recognition systems. *Image and Vision Computing*, 26, 2008.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110, June 2008. ISSN 1077-3142.

O Burchan Bayazit, Jyh-Ming Lien, and Nancy M Amato. Probabilistic roadmap motion planning for deformable objects. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 2126–2133. IEEE, 2002.

Matthew Bell. *Flexible object manipulation*. PhD thesis, DARTMOUTH COLLEGE Hanover, New Hampshire, 2010.

Matthew Bell and Devin Balkcom. Grasping non-stretchable cloth polygons. *The International Journal of Robotics Research*, 2009.

R. Bencina, M. Kaltenbrunner, and S. Jorda. Improved topological fiducial tracking in the reactivision system. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 99, june 2005. doi: 10.1109/CVPR.2005.475.

Christian Bersch, Benjamin Pitzer, and Soren Kammel. Bimanual robotic cloth manipulation for laundry folding. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1413–1419. IEEE, 2011.

Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL http://dx.doi.org/10.1109/34.121791.

R. C. Bose and D. K. Ray Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, March 1960. doi: 10.1016/S0019-9958(60)90287-4. URL http://dx.doi.org/10.1016/S0019-9958(60)90287-4.

Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

Duane C Brown. Reseau platen for cameras, April 17 1979. US Patent 4,149,788.

Joel Brown, Jean-Claude Latombe, and Kevin Montgomery. Real-time knot-tying simulation. *The Visual Computer*, 20(2-3):165–179, 2004.

Christopher R Calladine. Gaussian curvature and shell structures. *The Mathematics of Surfaces*, pages 19–26, 1986.

Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

Youngkwan Cho and Ulrich Neumann. Multi-ring color fiducial systems for scalable fiducial tracking augmented reality. In *Proc. of IEEE VRAIS*, page 212. Citeseer, 1998.

Philippe G Ciarlet. *Theory of shells*, volume 20. North Holland, 2000.

Christoffer Cromvik and Kenneth Eriksson. *Airbag Folding Based on Origami Mathematics*. Chalmers University of Technology, 2006.

Yunduan Cui, James Poon, Takamitsu Matsubara, Jaime Vails Miro, Kenji Sugimoto, and Kimitoshi Yamazaki. Environment-adaptive interaction primitives for human-robot motor skill learning. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 711–717. IEEE, 2016.

Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O'Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3893–3900. IEEE, 2011.

Erik D Demaine and Joseph O'Rourke. *Geometric folding algorithms*. Cambridge university press Cambridge, 2007.

Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):932–946, 2002.

Yuping Duan, Weimin Huang, Huibin Chang, Wenyu Chen, Jiayin Zhou, Soo Kng Teo, Yi Su, Chee Kong Chui, and Stephen Chang. Volume preserved mass–spring model with novel constraints for soft tissue deformation. *IEEE journal of biomedical and health informatics*, 20(1): 268–280, 2016.

Venketesh N Dubey and Jian S Dai. A packaging robot for complex cartons. *Industrial Robot: An International Journal*, 33(2):82–87, 2006.

Christof Elbrechter, Robert Haschke, and Helge Ritter. Bi-manual robotic paper manipulation based on real-time marker tracking and physical modelling. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1427–1432. IEEE, 2011a.

Christof Elbrechter, Robert Haschke, and Helge Ritter. Video for the conference paper: "bi-manual robotic paper manipulation based on real-time marker tracking and physical modelling", April 2011b. URL https://www.youtube.com/watch?v=-KrhjSB77-k.

Christof Elbrechter, Robert Haschke, and Helge Ritter. Folding paper with anthropomorphic robot hands using real-time physics-based modeling. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 210–215. IEEE, 2012a.

Christof Elbrechter, Robert Haschke, and Helge Ritter. Video for the conference paper: "folding paper with anthropomorphic robot hands using real-time physics-based modeling", November 2012b. URL https://www.youtube.com/watch?v=RxOZCkE6YRQ.

Jack Fastag. egami: Virtual paperfolding and diagramming software. In *Proceedings of 4th International Conference on Origami, Science, Mathematics and Education*, page 68, 2006.

Mark Fiala. ARTag, a fiducial marker system using digital techniques. *Proc. Computer Vision and Pattern Recognition*, 2, 2005. ISSN 1063-6919. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2005.74.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Jannik Fritsch and Sebastian Wrede. An integration framework for developing interactive robots. In *Software Engineering for Experimental Robotics*, pages 291–305. Springer, 2007.

Russell Gayle, Ming C Lin, and Dinesh Manocha. Constraint-based motion planning of deformable robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1046–1053. IEEE, 2005.

S. Gibbs, D. Tsichritzis, E. Casais, O. Nierstrasz, and X. Pintado. Class management for software communities, 1990.

Birgit Graf, Matthias Hans, and Rolf D Schraft. Care-o-bot ii—development of a next generation robotic home assistant. *Autonomous robots*, 16(2):193–205, 2004.

Birgit Graf, Ulrich Reiser, Martin Hägele, Kathrin Mauz, and Peter Klein. Robotic home assistant care-o-bot® 3-product vision and innovation platform. In *2009 IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 139–144. IEEE, 2009.

Alfred Gray, Elsa Abbena, and Simon Salamon. Modern differential geometry of curves and surfaces with mathematica. *AMC*, 10:12, 1998.

E. Grinspun. A discrete model of thin shells. In *ACM SIGGRAPH 2005 Courses*, page 4, 2005.

Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67. Eurographics Association, 2003.

Satyandra K Gupta, David A Bourne, KH Kim, and SS Krishnan. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338, 1998.

Dirk Haehnel, Sebastian Thrun, and Wolfram Burgard. An extension of the icp algorithm for modeling nonrigid objects with mobile robots. In *IJCAI*, volume 3, pages 915–920, 2003.

M Hans, B Graf, and RD Schraft. Robotic home assistant care-o-bot: Past-present-future. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 380–385. IEEE, 2002.

D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, (8):231–274, 1987.

L. He, Y. Chao, and K. Suzuki. A run-based two-scan labeling algorithm. *Image Processing, IEEE Transactions on*, 17(5):749–756, 2008.

Dominik Henrich and Heinz Wörn. *Robot manipulation of deformable objects*. Springer Science & Business Media, 2012.

David Hilbert, Stephan Cohn-Vossen, and Peter Nemenyi. *Geometry and the Imagination*. Chelsea Publishing Company New York, 1952.

Robert D. Howe and Yoky Matsuoka. Robotics for surgery. *Annual Review of Biomedical Engineering*, 1999.

Shouren Huang, Yuji Yamakawa, Taku Senoo, and Masatoshi Ishikawa. Robotic needle threading manipulation based on high-speed motion strategy using high-speed visual feedback. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4041–4046. IEEE, 2015.

Manfred Huber and Roderic Grupen. Learning to coordinate controllers–reinforcement learning on a control basis. In *15th International Joint Conference on Artificial Intelligence*, 1997.

David A. Huffman. Curvature and creases: A primer on paper. *Computers, IEEE Transactions on*, 100(10):1010–1019, 1976.

Thomas C. Hull. Configuration spaces for flat vertex folds. In *Proceedings of 4th International Conference on Origami, Science, Mathematics and Education*, page 361. A K Peters/CRC Press, 2006. doi: 10.1201/b10653. URL `http://dx.doi.org/10.1201/b10653`.

S. Hutchinson, G. D Hager, and P. I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996. ISSN 1042-296X.

Tetsuo Ida, Hidekazu Takahashi, Mircea Marin, Asem Kasem, and Fadoua Ghourabi. Computational origami system eos. In *Proceedings of 4th International Conference on Origami, Science, Mathematics and Education*, page 69, 2006.

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25 (2):328–373, 2013.

Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.

Shervin Javdani, Sameep Tandon, Jie Tang, James F O'Brien, and Pieter Abbeel. Modeling and perception of deformable one-dimensional objects. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1607–1614. IEEE, 2011.

Il-Kwon Jeong and Inho Lee. An Oriented Particle and Generalized Spring Model for Fast Prototyping Deformable Objects. In M. Alexa and E. Galin, editors, *Eurographics 2004 - Short Presentations*. Eurographics Association, 2004. doi: 10.2312/egs.20041014.

P Jiménez. Survey on model-based manipulation planning of deformable objects. *Robotics and computer-integrated manufacturing*, 28(2):154–163, 2012.

M. Kaltenbrunner. reacTIVision and TUIO: a tangible tabletop toolkit. In *Int. Conf. Interactive Tabletops and Surfaces*, 2009.

Hirokazu Kato and Mark Billinghurst. Marker tracking and HMD calibration for a Video-Based augmented reality conferencing system. In *Proc. 2nd Int. Workshop on Augmented Reality*, page 85, 1999. ISBN 0-7695-0359-4.

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

Fouad F Khalil and Pierre Payeur. Robotic interaction with deformable objects under vision and tactile guidance-a review. In *2007 International Workshop on Robotic and Sensors Environments*, pages 1–6. IEEE, 2007.

Fouad F Khalil and Pierre Payeur. *Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review*. INTECH Open Access Publisher, 2010.

Yasuhiro Kinoshita and Toyohide Watanabe. Estimation of folding operation using silhouette of origami. *IAENG International Journal of Computer Science*, 37(2), 2008.

Nishanth Koganti, Tomoya Tamei, Takamitsu Matsubara, and Tomohiro Shibata. Estimation of human cloth topological relationship using depth sensor for robotic clothing assistance. In *Proceedings of Conference on Advances In Robotics*, pages 1–6. ACM, 2013.

Kentaro Kokufuta and Tsutomu Maruyama. Real-time processing of local contrast enhancement on fpga. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 288–293. IEEE, 2009.

Maciej Kot. Mass spring models for the simulation of deformable objects. *Mass Spring Models for the Simulation of Deformable Objects*, 2014.

N Kothary, S Dieterich, JD Louie, DT Chang, LV Hofmann, and DY Sze. Percutaneous implantation of fiducial markers for imaging-guided radiation therapy. *Americal Journal of Roentgenology (AJR)*, 192(4), april 2009.

Shunsuke Kudoh, Tomoyuki Gomi, Ryota Katano, Tetsuo Tomizawa, and Takashi Suehiro. In-air knotting of rope by a dual-arm multi-finger robot. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6202–6207. IEEE, 2015.

Risto Kõiva. *Towards observable haptics: Novel sensors for capturing tactile interaction patterns*. PhD thesis, 2014.

Andrew M Ladd and Lydia E Kavraki. Motion planning for knot untangling. In *Algorithmic Foundations of Robotics V*, pages 7–23. Springer, 2004.

Karthik Lakshmanan, Apoorva Sachdev, Ziang Xie, Dmitry Berenson, Ken Goldberg, and Pieter Abbeel. A constraint-aware motion planning algorithm for robotic folding of clothes. In *Experimental Robotics*, pages 547–562. Springer, 2013.

Tung Ken Lam. Computer origami simulation and the production of origami instructions. In *Origami 4: Fourth International Meeting of Origami Science, Mathematics, and Education*, volume 4, page 237. AK Peters, 2009.

Florent Lamiraux and Lydia E Kavraki. Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research*, 20(3):188–208, 2001.

Robert J Lang. A computational algorithm for origami design. In *Proceedings of the twelfth annual symposium on Computational geometry*, pages 98–105. ACM, 1996.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Matt LeDuc, Shahram Payandeh, and John Dill. Toward modeling of a suturing task. In *Graphics Interface*, volume 3, pages 273–279. Citeseer, 2003.

Alex X Lee, Sandy H Huang, Dylan Hadfield-Menell, Eric Tzeng, and Pieter Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4402–4407. IEEE, 2014.

Alex X Lee, Max A Goldstein, Shane T Barratt, and Pieter Abbeel. A non-rigid point and normal registration algorithm with applications to learning from demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 935–942. IEEE, 2015a.

Alex X Lee, Henry Lu, Abhishek Gupta, Sergey Levine, and Pieter Abbeel. Learning force-based manipulation of deformable objects from multiple demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 177–184. IEEE, 2015b.

Qiang Li, Robert Haschke, and Helge Ritter. A visuo-tactile control framework for manipulation and exploration of unknown objects. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 610–615. IEEE, 2015.

Honghai Liu and Jian Dai. An approach to carton-folding trajectory planning using dual robotic fingers. *Robotics and Autonomous Systems*, 42(1):47–63, 2003.

Liang Lu and Srinivas Akella. Folding cartons with fixtures: A motion planning approach. *Robotics and Automation, IEEE Transactions on*, 16(4):346–356, 2000.

Ingo Lütkebohle, Julia Peltason, Lars Schillingmann, Britta Wrede, Sven Wachsmuth, Christof Elbrechter, and Robert Haschke. The curious robot-structuring interactive robot learning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4156–4162. IEEE, 2009.

Guilherme Maeda, Marco Ewerton, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Gerhard Neumann. Learning interaction for collaborative tasks with probabilistic movement primitives. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 527–534. IEEE, 2014.

Guilherme J Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks. *Autonomous Robots*, pages 1–20, 2016.

J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proc. ICRA*, 2010.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. *Unified simulation of elastic rods, shells, and solids*, volume 29. ACM, 2010.

Alejandro Marzinotto and Johannes A Stork. Rope through loop insertion for robotic knotting: A virtual magnetic field formulation. *arXiv preprint arXiv:1611.06070*, 2016.

James Massey. Shift-register synthesis and bch decoding. *Information Theory, IEEE Transactions on*, 15(1):122–127, 1969.

Jonathan Maycock, Daniel Dornbusch, Christof Elbrechter, Robert Haschke, Thomas Schack, and Helge Ritter. Approaching manual intelligence. *KI - Künstliche Intelligenz*, 24:287–294, 2010. ISSN 0933-1875.

Quan Miao, Guijin Wang, Chenbo Shi, Xinggang Lin, and Zhiwei Ruan. A new framework for on-line object tracking based on surf. *Pattern Recognition Letters*, 32(13):1564–1571, 2011.

Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4861–4868. IEEE, 2011.

Stephen Miller, Jur Van Den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

Jun Mitani. Recognition, modeling and rendering method for origami using 2d bar codes. In *Proceedings of 4th International Conference on Origami, Science, Mathematics and Education. 4OSME, Caltech, Pasadena CA*, 2006.

Daisuke Miyazaki, Saori Kagimoto, Masashi Baba, and Naoki Asada. Creating digital model of origami crane through recognition of origami states from image sequence. In *ACM SIGGRAPH ASIA 2010 Posters*, SA '10, pages 19:1–19:1, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0524-2. doi: 10.1145/1900354.1900375. URL `http://doi.acm.org/10.1145/1900354.1900375`.

F. Mokhtarian and R. Suomela. Robust image corner detection through curvature scale space. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(12):1376–1381, 1998. ISSN 0162-8828. doi: 10.1109/34.735812.

Mark Moll and Lydia E Kavraki. Path planning for minimal energy curves of constant length. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2826–2831. IEEE, 2004.

Mark Moll and Lydia E Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

T. Monahan. Bow making apparatus and method, November 16 1993. URL `https://www.google.com/patents/US5261578`. US Patent 5,261,578.

Alan Mutka, Damjan Miklic, Ivica Draganjac, and Stjepan Bogdan. A low cost vision based localization system using fiducial markers. *Proceedings of the 17th World Congress The International Federation of Automatic Control, Seoul, Korea*, 2008.

Ciro Natale. *Interaction control of robot manipulators: six degrees-of-freedom tasks*, volume 3. Springer Science & Business Media, 2003.

Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, volume 1, page 3, 2011.

Dinesh K Pai. Strands: Interactive simulation of thin solids using cosserat models. In *Computer Graphics Forum*, volume 21, pages 347–352. Wiley Online Library, 2002.

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

Peter Pastor, Mrinal Kalakrishnan, Franziska Meier, Freek Stulp, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):351–361, 2013.

Christoph J Paulus, Lionel Untereiner, Hadrien Courtecuisse, Stéphane Cotin, and David Cazier. Virtual cutting of deformable objects based on efficient topological operations. *The Visual Computer*, 31(6-8):831–841, 2015.

Antoine Petit, Vincenzo Lippiello, Giuseppe Andrea Fontanelli, and Bruno Siciliano. Tracking elastic deformable objects with an rgb-d sensor for a pizza chef robot. *Robotics and Autonomous Systems*, 88:187–201, 2017.

Jeff Phillips, Andrew Ladd, and Lydia E Kavraki. Simulated knot tying. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 841–846. IEEE, 2002.

BC Porter, DJ Rubens, JG Strang, J Smith, S Totterman, and KJ Parker. Three-dimensional registration and fusion of ultrasound and mri using major vessels as fiducial markers. *IEEE Transactions on medical imaging*, 20(4), april 2001.

William H Press, SA Teukolsky, WT Vetterling, and BP Flannery. Numerical recipes in c: the art of scientific computing, 994, 1992.

Anshuman Razdan and MyungSoo Bae. Curvature estimation scheme for triangle meshes using biquadratic bézier patches. *Computer-Aided Design*, 37(14):1481–1491, 2005.

Helge Ritter, Robert Haschke, and Jochen J Steil. A dual interaction perspective for robot cognition: grasping as a "rosetta stone". In *Perspectives of neural-symbolic integration*, pages 159–178. Springer, 2007.

David A Rosenbaum, Caroline M van Heugten, and Graham E Caldwell. From cognition to biomechanics and back: The end-state comfort effect and the middle-is-faster effect. *Acta psychologica*, 94(1):59–85, 1996.

E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, Nov 2011. doi: 10.1109/ICCV.2011.6126544.

Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

Mitul Saha and Pekka Isto. Motion planning for robotic manipulation of deformable linear objects. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2478–2484. IEEE, 2006.

Mitul Saha and Pekka Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

Matthias Schöpfer, Carsten Schürmann, Michael Pardowitz, and Helge Ritter. Using a piezo-resistive tactile sensor for detection of incipient slippage. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–7. VDE, 2010.

Matthias Schroder, Christof Elbrechter, Jonathan Maycock, Robert Haschke, Mario Botsch, and Helge Ritter. Real-time hand tracking with a color glove for the actuation of anthropomorphic robot hands. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 262–269. IEEE, 2012.

John Schulman, Ankush Gupta, Sibi Venkatesan, Mallory Tayson-Frederick, and Pieter Abbeel. A case study of trajectory transfer through non-rigid registration for a simplified suturing scenario. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4111–4117. IEEE, 2013a.

John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. 2013b.

John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. Learning from demonstrations through the use of non-rigid registration. In *Robotics Research*, pages 339–354. Springer, 2016.

G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE transactions on pattern analysis and machine intelligence*, page 2024–2030, 2006. ISSN 0162-8828.

Shadow Robot Company. The Shadow Dextrous Hand, a. URL `http://www.shadowrobot.com/hand/overview.shtml`.

Shadow Robot Company. The Shadow Motor Hand, b. URL `https://www.shadowrobot.com/tag/motor-hand`.

Hiroshi Shimanuki, Jien Kato, and Toyohide Watanabeuki. Construction of 3d virtual origami models from sketches. In *Origami 4: Fourth International Meeting of Origami Science, Mathematics, and Education*, volume 4, page 237. AK Peters, 2009. doi: 10.1201/b10653. URL `http://dx.doi.org/10.1201/b10653`.

Martin W Short and James H Cauraugh. Planning macroscopic aspects of manual control: End-state comfort and point-of-change effects. *Acta Psychologica*, 96(1):133–147, 1997.

P. Sobrevilla and E. Montseny. Fuzzy sets in computer vision: an overview. *Mathware & Soft Computing*, 10:71–83, 2003.

Guang Song and Nancy M Amato. A motion-planning approach to folding: from paper craft to protein folding. *Robotics and Automation, IEEE Transactions on*, 20(1):60–71, 2004.

Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. ISBN 049508252X.

Jan Steffen, Christof Elbrechter, R. Haschke, and Helge J. Ritter. Bio-inspired motion strategies for a bimanual manipulation task. In *Int. Conf. on Humanoid Robots*, 2010.

John M Sullivan. Curvature measures for discrete surfaces. In *ACM SIGGRAPH 2005 Courses*, page 3. ACM, 2005.

T. Tachi. Simulation of rigid origami. *Origami 4*, page 175, 2009.

Kenta Tanaka, Yusuke Kamotani, and Yasuyoshi Yokokohji. Origami folding by a robotic hand. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2540–2547. IEEE, 2007.

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 205–214, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: 10.1145/37401.37427. URL `http://doi.acm.org/10.1145/37401.37427`.

Lukas Twardon and Helge Ritter. Interaction skills for a coat-check robot: Identifying and handling the boundary components of clothes. pages 3682–3688. IEEE, 2015a. doi: 10.1109/ICRA.2015.7139710.

Lukas Twardon and Helge Ritter. Interaction skills for a coat-check robot: Identifying and handling the boundary components of clothes. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3682–3688. IEEE, 2015b.

André Ückermann, Robert Haschke, and Helge Ritter. Realtime 3d segmentation for human-robot interaction. pages 2136–2143, 2013.

Jur Van Den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. In *Algorithmic Foundations of Robotics IX*, pages 409–424. Springer, 2010.

Jur Van Den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. In *Algorithmic Foundations of Robotics IX*, pages 409–424. Springer, 2011.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

D. Wagner and D. Schmalstieg. Artoolkitplus for pose tracking on mobile devices. In *Proc. of 12th Computer Vision Winter Workshop (CVWW'07)*, pages 139–146, 2007.

D. Wagner and D. Schmalstieg. Making augmented reality practical on mobile phones, part 1. *Computer Graphics and Applications, IEEE*, 29(3):12 –15, may-june 2009. ISSN 0272-1716. doi: 10.1109/MCG.2009.46.

Hidefumi Wakamatsu, Akira Tsumaya, Eiji Arai, and Shinichi Hirai. Manipulation planning for knotting/unknotting and tightly tying of deformable linear objects. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2505–2510. IEEE, 2005.

Fei Wang, Etienne Burdet, Vuillemin Ronald, and Hannes Bleuler. Knot-tying with visual and force feedback for vr laparoscopic training. In *Proc. of 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE-EMBS)*, number LSRO-CONF-2007-023, 2005.

Ping Chuan Wang, Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Perception for the manipulation of socks. In *IROS*, pages 4877–4884, 2011.

Sheng-Feng Wang, Huai-Yi Hsu, and An-Yeu Wu. A very low-cost multi-mode reed-solomon decoder based on peterson-gorenstein-zierler algorithm. In *Signal Processing Systems, 2001 IEEE Workshop on*, pages 37–48. IEEE, 2001.

Erik Weitnauer, Robert Haschke, and Helge Ritter. Evaluating a physics engine as an ingredient for physical reasoning. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 144–155. Springer Berlin Heidelberg, 2010.

J. Wienke and S. Wrede. A middleware for collaborative research in experimental robotics. In *System Integration (SII), 2011 IEEE/SICE International Symposium on*, pages 1183 –1190, dec. 2011. doi: 10.1109/SII.2011.6147617.

Jay Windley. Clavius photography crosshairs – "what are those little crosshairs in apollo photographs?". URL `http://www.clavius.org/photoret.html`.

Markus Windolf, Nils Götzen, and Michael Morlock. Systematic accuracy and precision analysis of video motion capturing systems – exemplified on the vicon-460 system. *Journal of Biomechanics*, 41(12):2776 – 2780, 2008. ISSN 0021-9290. doi: 10.1016/j.jbiomech.2008.06.024.

Yuji Yamakawa, Akio Namiki, Masatoshi Ishikawa, and Makoto Shimojo. One-handed knotting of a flexible rope with a high-speed multifingered hand having tactile sensors. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 703–708. IEEE, 2007.

Yuji Yamakawa, Akio Namiki, and Masatoshi Ishikawa. Dynamic high-speed knotting of a rope by a manipulator. *International Journal of Advanced Robotic Systems*, 10(10):361, 2013.

Yang Yang, Qixin Cao, Charles Lo, and Zhen Zhang. Pose estimation based on four coplanar point correspondences. In *Proc. of the 6th int. conference on Fuzzy systems and knowledge discovery - Volume 5*, pages 410–414, Tianjin, China, 2009. IEEE Press. ISBN 978-1-4244-4545-5.

Wei Yao, Ferdinando Cannella, and Jian S Dai. Automatic folding of cartons using a reconfigurable robotic system. *Robotics and Computer-Integrated Manufacturing*, 27(3):604–613, 2011.

E. Yoshida, K. Ayusawa, I. G. Ramirez-Alpizar, K. Harada, C. Duriez, and A. Kheddar. Simulation-based optimal motion planning for deformable object. In *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 1–6, June 2015. doi: 10.1109/ARSO.2015.7428219.

Zhong You and Kaori Kuribayashi. Expandable tubes with negative poisson's ratio and their application in medicine. In *Origami4: Fourth International Meeting of Origami Science, Mathematics, and Education*, pages 117–127. Citeseer, 2006.

Zhengyou Zhang. Iterative point matching for registration of free-form curves. 1992.

Mianlun Zheng, Qing Guo, Qi Wang, Huasong Han, Zhiquan Hu, and Zhiyong Yuan. Geometry based rope knot tying simulation. In *International Conference on Logistics Engineering, Management and Computer Science (LEMCS 2014)*. Atlantis Press, 2014.

Feng Zhou, H.B.-L. Duh, and M. Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, pages 193 –202, sept. 2008. doi: 10.1109/ISMAR.2008.4637362.

You Zhou, Martin Do, and Tamim Asfour. Coordinate change dynamic movement primitives—a leader-follower approach. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5481–5488. IEEE, 2016.