Die approbierte Originalversion dieser Dissertation ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

DISSERTATION

Object modelling for cognitive robotics

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
E376
Institut für Automatisierungs- und Regelungstechnik

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. Thomas Mörwald
geb. am 16.05.1982
Matr. Nr.: 0255334
Reithlegasse 15/3
A-1190 Wien

Wien, im April 2013

Thomas Mörwald

Abstract

The development of robots received great attention in the last decades. Progressing from hard-coded pick and place operations common for industrial applications, the need for more intelligent solutions emerged. The field of cognitive robotics evolved, where tasks are no more hard-coded processes that are executed monotonously. The concept of intelligent robots, known form science fiction, found its way into science, where soon methods appeared that allow for reasoning, representing knowledge, interacting with humans and so forth. In this thesis the focus lies on cognitive perception, that allows to learn and reason about objects as the robot perceives them. The importance of never-ending learning methods, the ability to handle partial information and fusion of knowledge from different cues for a better understanding of the appearance and properties of objects is demonstrated.

The appearance of objects is given by their shape and colour. Geometric models, such as B-spline curves and surfaces are used to segment range images and simultaneously reconstruct the shape of smooth, continuous surface patches. These patches are grouped to objects according to relations inspired by Gestalt principles. Colour information is mapped onto the shape, resulting in a model for the appearance of the object for a single view.

The key for learning and reasoning is to identify objects that have already been learned and to assign new information to them. In the context of perception that is, to visually track it and to self-evaluate observations to distinguish good and bad sensor data (e.g. sensor noise, occlusions, reflections, and so forth). For visual tracking the previously reconstructed appearance of the object is used. With respect to a prior pose, a Monte Carlo particle filter (MCPF) evaluates various pose hypothesis, efficiently following the object movement, including rigid 3D translations and rotations. A novel algorithm for evaluation of the tracking state, called tracking-state-detection (TSD), is proposed which allows to reason about the tracking quality, detects whether tracking is valid or lost, or if the object is occluded.

The TSD allows to identify new good views, from which new information can be used to extend existing appearance models, but also to learn about the physical behaviour of objects. The trajectory of an object under robotic manipulation is observed, where a robotic finger is pushing it. This allows to learn or extend a probabilistic motion model. That is, to assign probabilities between coordinate frames attached to the object, the robotic finger and the environment. The advantage of a probabilistic physical model is, in contrast to Newtonian mechanics, that it allows for generalisation to new object shapes and pushing configurations. This makes it perfectly suitable for cognitive robotics. Furthermore physical predictions can be used as prior poses for tracking, increasing accuracy and robustness especially in difficult situations (e.g. motion blur during fast movement, partial or full occlusion).

This manuscript is best to read in colour.

Acknowledgement

I would like to sincerely thank all people who supported me while working on this thesis. My foremost thanks go to my supervisor Markus Vincze and mentor Michael Zillich. I am grateful for the inspiration, freedom and support they provided me. I would also like to thank my external reviewer Aleš Leonardis, Professor at the University of Birmingham and the University of Ljubljana for his encouragements.

I am grateful for all the colleagues who accompanied me during my work. Especially I want to thank Johann Prankl, Andreas Richtsfeld, Marek Kopicki and Jonathan Balzer for all the inspiring, passionate discussions and debates we had about so many topics, leading to many valuable publications.

My gratitude also goes to my parents Johann and Anna Mörwald, who guided me towards a scientific career by exciting my curiosity and supporting me at difficult hurdles. And especially warm thanks to my beloved partner and dear friend Yvonne Seiler, who inspired me to look at various problems from a different point of view.

Contents

Nomenclature IV

1 Introduction 1
- 1.1 Problem statement: modelling objects . . . 3
- 1.2 Outline and contributions . . . 6

2 Segmentation and reconstruction 11
- 2.1 Related work . . . 12
- 2.2 B-spline fitting . . . 13
  - 2.2.1 Definition of B-spline curves and surfaces . . . 14
  - 2.2.2 Fitting curves to 2D point-clouds . . . 16
  - 2.2.3 Surface fitting . . . 25
- 2.3 Surface segmentation . . . 28
  - 2.3.1 Pre-segmentation . . . 28
  - 2.3.2 Model selection . . . 29
- 2.4 Object segmentation . . . 30
  - 2.4.1 Grouping of parametric surfaces . . . 30
- 2.5 Learning . . . 31
  - 2.5.1 Object registration . . . 31
  - 2.5.2 Surface merging . . . 32
- 2.6 Results . . . 34
  - 2.6.1 B-spline curve fitting . . . 34
  - 2.6.2 Surface segmentation . . . 40
  - 2.6.3 Object segmentation . . . 43
- 2.7 Discussion . . . 45

3 Colour based object tracking 47
- 3.1 Related work . . . 49
- 3.2 Pose estimation . . . 50
  - 3.2.1 Transformations on the Euclidean group . . . 50
  - 3.2.2 Monte Carlo particle filtering (MCPF) . . . 51
  - 3.2.3 Image processing and confidence evaluation . . . 54
- 3.3 Learning . . . 56
  - 3.3.1 Tracking-state-detection (TSD) . . . 56

3.3.2 Texture mapping . . . 58
3.3.3 SIFT mapping and object re-detection . . . 59
3.3.4 Model completeness . . . 60
3.4 Results . . . 62
3.4.1 Evaluation of the tracking error . . . 62
3.4.2 Accuracy and precision . . . 63
3.4.3 Robustness . . . 64
3.4.4 Performance . . . 65
3.4.5 Video . . . 65
3.5 Discussion . . . 67

4 Physical prediction and robotic manipulation **69**
4.1 Related work . . . 70
4.2 Prediction . . . 72
4.3 Tracking . . . 74
4.4 Learning . . . 75
4.5 Results . . . 76
4.5.1 Tracking . . . 76
4.5.2 Prediction . . . 80
4.6 Discussion . . . 86

5 Conclusion **87**
5.1 Summary . . . 87
5.2 Outlook . . . 88

Bibliography **91**

Curriculum Vitae **99**

Nomenclature

Segmentation and Reconstruction

S	... savings of a surface model hypothesis
$\kappa_{0,1,2}$	... MDL weights
Ξ	... knot vector
ξ_i	... element of the knot vector
ξ	... parameter of a B-spline curve
u, v	... parameters of a B-spline surface
Ω	... parameter space of a B-spline curve or surface
$N_{i,p}$	... i-th B-spline basis function of p-th order
$M_{j,p}$	... j-th B-spline basis function of p-th order
$\mathbf{c}$	... B-spline curve
$\mathbf{b}$	... control vector
$\mathbf{S}$	... B-spline surface
$\mathbf{B}$	... control grid
$\mathbf{p}$	... point
$\mathbf{n}$	... normal vector
$\mathbf{t}$	... tangent vector
$\mathbf{o}$	... outward pointing normal vector
e	... error function
f	... objective function
w	... weighting factor or -function
d	... signed distance to B-spline curve
ρ	... curvature of the B-spline curve
σ	... standard deviation or transition width
ε	... threshold
$\mathbf{f}$	... force vector
$\mathbf{K}$	... stiffness matrix
$\mathbf{R}$	... regularisation matrix

Tracking and Robotic Manipulation

t ... time steps $t \in \mathbb{N} \cup \{0\}$
I ... colour image
$\mathbf{T}$ ... rigid transformation (rotation, translation)
$\mathbf{t}$ ... translation
$\mathbf{R}$ ... rotation (matrix form)
$\mathfrak{i}, \mathfrak{j}, \mathfrak{k}$ ... imaginary units
$\mathfrak{q}$ ... quaternion
r ... real value of quaternion
$\theta_{x,y,z}$ ... imaginary value of quaternion
$\boldsymbol{\theta}$ ... vector of imaginary values of quaternion, i.e. rotation
$\mathbf{x}$ ... state vector of a particle
$\tilde{\mathbf{x}}$ ... posterior particle
$\mathcal{N}$ ... normal distributed noise
σ ... standard deviation
c ... confidence
p ... probability
P ... discrete probability
i ... particle index
N ... number of particles
w ... importance weight
$\mathbf{y}$ ... observation
$\mathbf{g}$ ... colour gradient
M ... object model projected to image space
m ... match value
s ... normalizing factor
h ... hue value of the HSV colour space
$\delta(\mathbf{x})$ ... delta-Dirac mass located in $\mathbf{x}$
u, v ... pixel coordinates in image space
$\mathbf{X}_t^f$ ... set of fixed particles
o ... detection success
$\mathbf{q}_k$ ... fixed point on the object surface
e ... tracking error
A, B, O ... coordinate frames

Chapter 1

Introduction

For an intelligent robot it is desirable to operate in an unknown environment, where the robot will encounter new situations and objects. E.g. the robot will navigate in rooms and areas it has not been before and it will perceive and interact with novel objects. A typical application area is service robotics, which is concerned with systems that are designed to support humans in home or office environments. Since it is neither possible to provide information about every home and office nor about all the objects encountered there, it is necessary to equip the robot with the ability to learn things from scratch. Of course many constraints can still be taken into account in advance but these are rather general (e.g. objects rest on a supporting surface; the field of gravity is pointing downwards) or derived from the robotic system itself, like from the embodiment. This means that knowing very little, the robot should be able to create models about the world and to update them as soon as new information is gathered. In the context of robotics these abilities characterise the field of *cognitive robotics*. To quote from [55]:

> “In its most general form, we take *cognitive robotics* to be the study of the knowledge representation and reasoning problems faced by an autonomous robot (or agent) in a dynamic and incompletely known world.”

Disassembling this sentence yields the following subtasks:

- *Knowledge representation:* Represent knowledge that allows for extension and modification over time.
- *Reasoning:* Reason about the world using and updating the current knowledge.
- *Autonomous robot:* Integrate knowledge about the embodiment of the robot and consider its possibilities.
- *Dynamic world:* Detect and observe changes in the environment as well as in the present state of knowledge.
- *Incomplete world:* Identify knowledge gaps, i.e. missing information.

These tasks strongly depend on each other and in fact they are usually highly integrated. However, since the items of the list are describing modules in a very generic way it is hard to integrate them as functions or algorithms in a robotic system. This results from the fact that one big piece is missing in the quotation above, namely the *purpose* of a cognitive robot. A nice and general way describing this in the context of cognitive robotics is stated in the work of Levesque and Reiter [56]:

> "With respect to robotics, our goal (like that of many in AI) is high-level robotic control: develop a system that is capable of generating actions in the world that are appropriate as a function of some current set of beliefs and desires. What we do not want to do is to simply engineer robot controllers that solve a class of problems or that work in a class of application domains."

Therefore the purpose of the robot is what we want it to belief and motivate. Consider the example of two home robots and their owner who wants them to bring her/him a cornflakes box. One robot is equipped with a hard-coded function which is executed exactly the same way each time the owner calls it. Figure 1.1 shows a simplistic example of a hard-coded task execution.

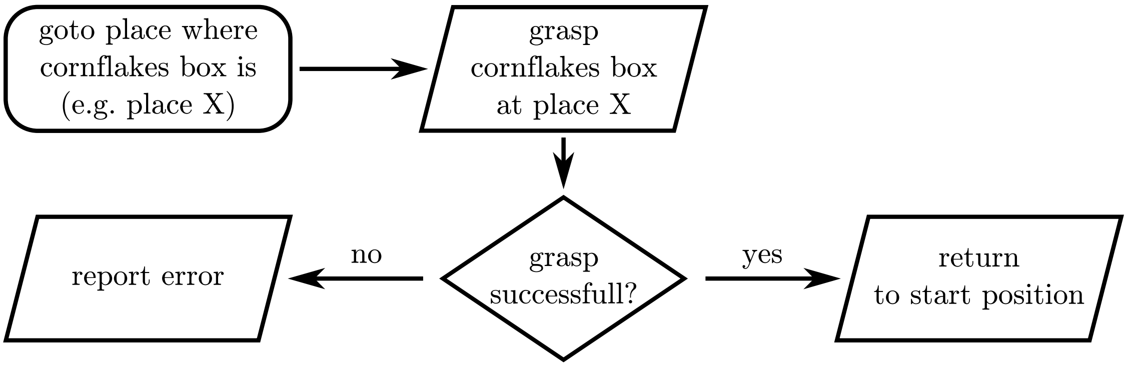

goto place where cornflakes box is (e.g. place X)
grasp cornflakes box at place X
grasp successfull?
no
report error
yes
return to start position

Figure 1.1: Hard-coded task execution for bringing a box of cornflakes.

The other robot is a cognitive one, where the command "Bring me the cornflakes box!" is transformed into a motive considering the knowledge of the robot represented as beliefs (e.g. probabilities of a cornflakes box being at a certain location). Figure 1.2 shows a simplistic example of a cognitive robot.

Obviously the first robot will struggle once the environment changes, e.g. when the cornflakes box is placed at a different location. The second one will eventually succeed because it considers a changing environment and updates its knowledge about it during execution time. The example demonstrates the importance of a cognitive understanding of the world for a flexible operation of robots. That is, constantly updating its knowledge, identifying gaps therein which ideally results in the motive to fill them. In the example above such a gap would pop out when the robot is commanded to bring something it has never seen before. This would trigger the

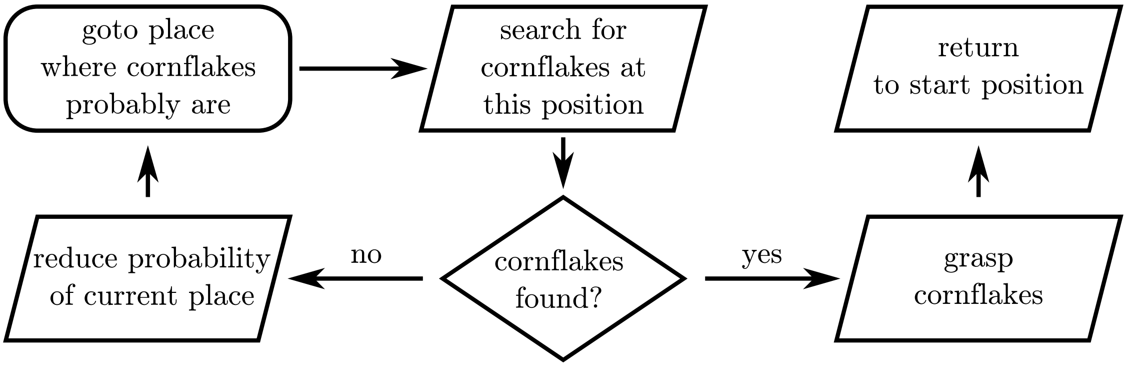

goto place where cornflakes probably are
search for cornflakes at this position
return to start position
reduce probability of current place
no
cornflakes found?
yes
grasp cornflakes

Figure 1.2: Cognitive task execution for bringing a box of cornflakes, taking into account beliefs about the world.

motive to learn about that *thing*, which from now on we refer to as *object*. A common way to do that is tutor-driven learning [51, 91, 92], where a tutor shows the object to the robot. To perform tasks including it, the robot will gather information about the object

- appearance including shape and colour, and its
- function including the physical behaviour.

Ideally these properties are generalizable over object instances. Knowing the typical appearance of a mug in general rather then only a few specific examples, would allow the robot not only to bring the ones known, but any. And analogously, generalization allows to perform tasks on object classes rather than instances (e.g. detection). Therein lies the real challenge for cognitive robotics. To equip the robot with algorithms that are able to learn the features above such that they describe the object in a meaningful way.

1.1 Problem statement: modelling objects

We place this thesis in the field of cognitive vision. That is to perceive the world with image- and additionally depth sensors, taking into account the context of a cognitive robot as described above. The term *cognitive vision* is often used to indicate a cognitively inspired method. We rather refer to it as image analysis that outputs meaningful features for the robot. Further we focus on the scenario of tutor-driven or self-motivated learning. In more detail, a number of unknown objects are placed in front of the robot which then builds up models for *shape*, *colour* and *physical behaviour*. In summary, the goal is to learn arbitrary unknown objects. That is to build up the knowledge of the robot using models that allow to successively improve and refine. For a cognitive robot system the requirements for such models are as follows.

Shape is often modelled using triangle meshes, which are easy to compute but do not convey much meaning. A more sophisticated way is to partition the object into smooth, continuous surfaces and select higher order polynomials to approximate those. This incorporates meaningful features like boundaries (i.e. edges), curvatures, assembly and polynomial orders of the surfaces. Hence it is much easier to generalize specific objects to classes. E.g. a mug could be described as a cylinder of a certain size with one side open and the other one closed. The left of Figure 1.3 shows the partial shape model of a cylindrical object, consisting of trimmed B-spline surfaces.

Colour information, captured by a camera is usually modelled as mixture of red, green and blue (RGB). A strong cue is provided by the gradients of a colour image since colour itself strongly depends on the lighting conditions. However, for pixel-wise comparison gradients are not very distinctive. A more suitable way is to use histograms of colour and gradients of image patches. Popular patch descriptors using histograms are SIFT [61] and SURF [3]. To achieve a cognitive model these descriptors require to be modifiable and extendible and, to link them with the object, be attached to the surface. This linkage incorporates the understanding of when a certain descriptor is clearly visible, distorted or occluded. Once a model of such shape dependent descriptor is built, it is straight forward to recognize and estimate the pose of the object in the future [72]. Figure 1.3 (right) shows colour information (i.e. texture) mapped onto the B-spline surfaces of the shape model.

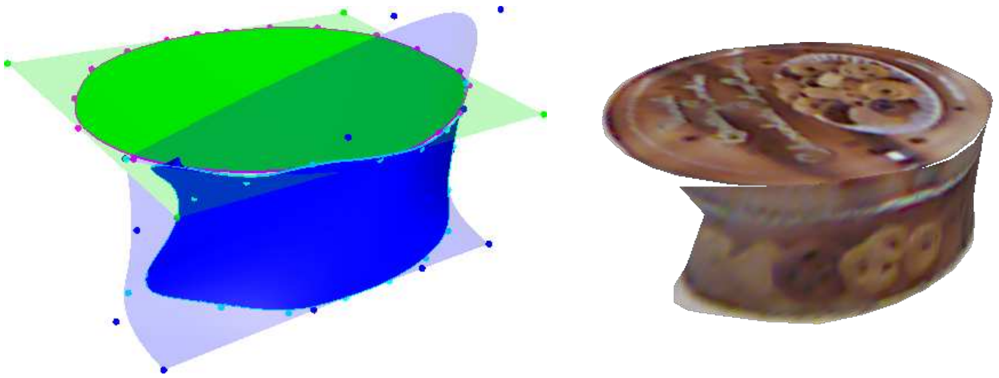


Figure 1.3: Models for the object appearance. Trimmed B-spline surface patches represent the 3D shape of the object (left). Colour information is mapped to the surface model (right).

Physical behaviour is typically implemented using Newtonian mechanics. Considering a cognitive system, parameters like friction, object dimension and mass are updated whenever new information is gathered. However, in the Newtonian representation the structure of the shape itself is hard to update. Consider an egg which, as a first guess, is modelled by a sphere. After several observations during which physical parameters are learned the visual system recognizes that an ovoid (egg shape) represents the shape in a better way. Knowing that the physical behaviour of these two primitives are significantly different, and the physical parameters depend strongly on the shape, the information already learned is worthless. Even if we consider an ovoid in the first place the physical predictions based on Newtonian mechanics do not match reality, although they may look plausible. Thus a more general representation, namely a probabilistic is preferred. Therein the behaviour of the object under a certain configuration of action is estimated using probabilities (e.g. an upright object is more likely to topple when being poked than a flat lying one). This representation implicitly models the geometry of the object but, in contrast to the Newtonian mechanics approach, allows smooth transition between different shapes. Figure 1.4 shows the location of coordinate frames of interacting objects. Probabilities, assigned to the space of all possible transformations between these coordinate frames, encodes the physical behaviour of the object. This allows to predict correct physical behaviour even if the geometry is not fully given.

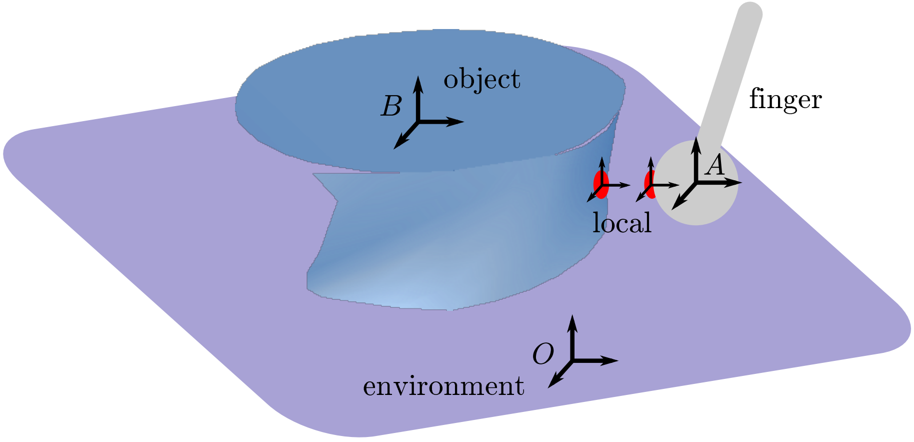


Figure 1.4: Model for the physical behaviour. Probabilities assigned to the transformations between the reference frame of the *object*, *robot finger*, *environment* and the *local* contact frames encode the physical behaviour.

These representations, modelling beliefs that change and extend over time, require methods that are able to cope with incompleteness. In other words, algorithms should not require a full model for them to work. This implies that object models are created in parallel rather then step by step. Of course one cannot assign colour if no region or shape defining the object is given. But once parts of it are known also colour and physical behaviour are learned. And with every new information arriving (e.g. new point of view) the models are extended and refined. In cognitive robotics this corresponds to the motive to learn about the world. In terms of Figure 1.5, our framework segments an object and creates partial models of the shape and colour to track and detect it. Then it continuously reflects the current models and updates them for a better understanding. A positive side effect thereof is, that using all the information gathered so far typically increases robustness, as will be shown in the subsequent chapters.

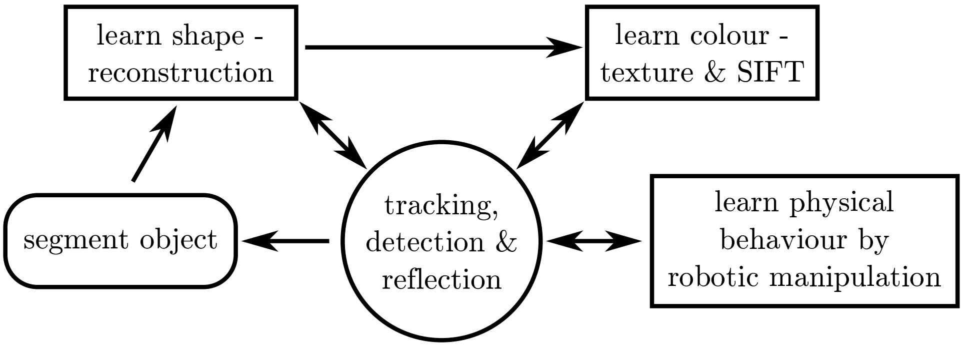

learn shape - reconstruction
learn colour - texture & SIFT
segment object
tracking, detection & reflection
learn physical behaviour by robotic manipulation

Figure 1.5: Cognitive task execution for learning shape, colour and physical behaviour. First an object of interest is segmented and the shape is learned. Then colour information is attached to the shape model which allows for visual tracking and detection. Reflecting its beliefs the robot selects new situations (i.e. point of view) and triggers the algorithms for updating the models.

1.2 Outline and contributions

This thesis presents a system that learns objects from scratch using colour and depth perception. Our contribution lies in the development of methods and algorithms that create models for the *shape*, *colour* and *physical behaviour* of the object in a cognitive sense. In other words, we use knowledge representations that allow for cognitive understanding of the world and that extend and modify over time. Following the scheme of Figure 1.5, this thesis proceeds as follows. First, the object is segmented from the sensor data and its shape is reconstructed. Since we are using the same shape models for both, segmentation and reconstruction, this is done in a simultaneous manner as will be shown in Chapter 2. Second, we project the shape model from the first view into image space and attach colour information accordingly. This textured model is used for tracking and further for detection with SIFT features, whenever the tracking algorithm looses the object. The methods developed for this are presented in Chapter 3. Third, employing our tracking algorithm and using the shape model as depicted in Figure 1.4, Chapter 4 presents the probabilistic formulation for predicting the physical behaviour and how it is learned by applying simple robotic manipulation.

Segmentation and reconstruction (Chapter 2)

Starting from raw depth sensor data, first planar approximations are computed which are further refined using higher order surfaces. Inspired by the principles of Gestalt [100] appearance features such as curvature, normals direction, proximity, colour and texture are used to group these surfaces. The output of this module are object models composed of trimmed B-spline surfaces. The latter are parametric free form surfaces with meaningful features like polynomial degree, curvature, normals and boundary edges (modelled with B-spline curves). Defining those surfaces in the image plane of the range sensor increases the efficiency of the algorithm and allows for a direct mapping from 2D image points to 3D surface points and vice versa. In [83] we *segment unknown objects* typical for indoor environments. We present a 3D perceptual grouping approach of B-spline surface patches of various degrees of freedom to accurately and robustly segment objects. In [73] we use *trimmed B-spline surfaces for range image segmentation and data abstraction.* We propose to define the parametric domain of the surfaces in the image space for two reasons. First, we can define trimming curves (i.e. B-spline curves) to smooth the segmentation boundaries and align them to image edges leading to a higher accuracy of the segmentation. Second, we can directly map 2D pixels to 3D surface points and vice versa leading to a trivial assignment of pixel information to surface points. Further the convex hull property of B-spline curves increases the performance and robustness of finding pixels being part of a certain segment. The property states that a B-spline curve completely lies within the convex hull of its control points. Therefore points outside this convex hull do not belong to the respective segment. The choice for B-spline surfaces was inspired by our recently published reconstruction approach exploiting isogeometric finite-elements methods for variational reconstruction tasks in vision [2].

Contribution: The output of the algorithm developed in this chapter is an object model consisting of trimmed B-spline surfaces. Each of them is modelling a smooth continuous part of the object. The model selection scheme provides the best fitting B-spline surfaces, constructing the shape of the object such that it fulfils the requirements mentioned in Section 1.1. I.e. the boundaries are well defined and the normal, curvature, order and type of surface is given at any point. The work on fitting B-spline models was integrated into the point-cloud library[1], which received enormous attention in the last years.

Colour based object tracking (Chapter 3)

Given the shape and colour of one view of the object we want to add information about missing parts to the model. Considering the tutor-driven scenario, the object is moved in front of the robot which requires to visually track it. While turning it, the robot adds previously occluded parts and replaces areas with better information quality both for shape and colour. This requires a qualitative statement of the current tracking state. Hence we extended a common approach for visual tracking, namely Monte Carlo particle filtering, by a method called *tracking-state-detection* (TSD) [74]. This models a qualitative understanding of the world (e.g. “This view is a good one.”) but also of the robot itself (e.g. “I am still tracking correctly.”). In our framework, TSD is employed for three tasks. First for identifying views which are suitable for learning, for which we developed a novel probabilistic formulation for *self-evaluation and prediction* of object poses [105]. Second for triggering reinitialization, using the present object model once tracking is lost, which was published as part of the *Blocks World Robotic Vision Toolbox* (BLORT) in [72]. Third for evaluation of the observation quality of a certain trajectory [48], where we *learn to*

---

[1] `www.pointclouds.org/blog/trcs/moerwald`

predict how rigid objects behave under simple manipulation. To improve tracking robustness, we take advantage of *colour texture* and propose an *improved functional for evaluating the particle weights* to avoid locking at difficult poses [75]. Furthermore we introduce *iterative particle filtering*, also called *recursive particle filtering* to the community, which significantly improves the performance.

Contribution: When developing the methods for this thesis, we found that the key ingredient for creating meaningful object models is the knowledge of what is happening during learning. Therefore we developed algorithms that visually track the object and detect it whenever it got lost. More importantly, we presented a method that allows to reason about the current tracking status. As this issue was and still is not sufficiently discussed by the community we found it to be the main reason why a lot of sophisticated tracking approaches do not pave the way to real world robotic applications. Our highly robust solution for tracking objects and determining the state of tracking in a qualitative manner did so (ROS module BLORT[2], PAL-robotics[3], CogX project[4]).

Physical prediction and robotic manipulation (Chapter 4)

The physical behaviour of the object is learned in a probabilistic framework. By pushing and poking with a robotic arm and considering the model gathered so far, the robot observes the physical effect, i.e. movement, of the object. Probabilities are assigned to certain arm-object-environment configurations over time. This allows to predict and constrain the object movement, increasing the robustness of tracking and grasping. Furthermore the probabilistic model leads to a generalizable understanding of the physical world where no understanding of Newtonian mechanics and its sensitive parameters are required. In a collaboration with the University of Birmingham a physical prediction system based on a probabilistic representation [47, 45] was exploited as follows. First the probabilistic predictors using the visual observations from our pose tracking algorithm [48] were trained. Then, once the physical model is learned, we increase the robustness of the tracking algorithm by predicting the object pose for the next observation [71]. This allows the system to *predict the object poses where it is fully occluded, out of view, or affected by strong noise.* Further a motion prediction system based on Newtonian mechanics was tested for tracking [25]. Even though the physical parameters used were estimated to best fit real training data and plausible predictions were obtained, slight deviations in the test data led to strong deviations.

Contribution: Physical models that meet the requirements as mentioned in Section 1.1 are barely studied, especially when dealing with 6 degree-of-freedom movement and relying on partial shape models. The proposed methods provide the essen-

---

[2] `www.ros.org/wiki/blort`

[3] `www.pal-robotics.com/blog/internships-at-pal-robotics-bence-magyar-reports`

[4] `http://cogx.eu/download`

tial abilities for a robot to operate in an unknown environment, where other methods completely fail as we will show in Chapter 4.

Conclusion (Chapter 5)

In Chapter 5 we summarize the results achieved and give an outlook on work in progress and possible improvements for the future.

Chapter 2

Segmentation and reconstruction

An important domain of mobile robotics is perception, *the extraction of meaning from sensory input*, allowing autonomous robots to operate in complex environments performing user-defined tasks. Sarkar and Boyer [90] introduced a classificatory structure for perceptual organisation in computer vision, showing different levels of abstraction to bring meaning to the data under the assumption that our world is not visually chaotic, but has structure and organisation.

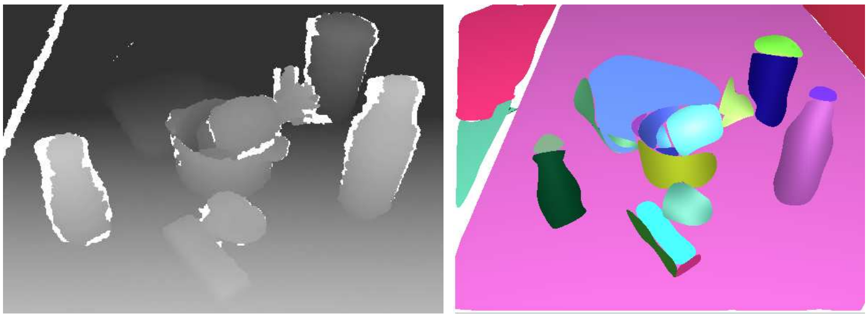


Figure 2.1: Segmentation of an range image captured by sensors such as the Microsoft Kinect. Left: Output of the sensor with missing data points (white). Right: Parametric models like planes and B-spline surfaces trimmed by curves.

Data abstraction brings advantages for further processing: Representing data as parametrized models, as shown in Figure 2.1, means data reduction and allows to establish a continuous model with distinctive characteristics such as its polynomial order, area, curvature and normals. This model is more accurate than the raw sensor data, because it incorporates prior knowledge of the world. Relations can be defined between higher level entities beyond raw point clouds (e.g. the finger tip of a robotic hand touching the surface of an object). Knowing parameters such as the surface normal and curvature allows to reason about grasping behaviour and stability. Furthermore, physical behaviour constrained by these relations can be incorporated [47] and even learned when model based object tracking algorithms [71, 25] observe the motion of a certain object during a defined robotic manipulation [48].

Overview: Section 2.1 gives an overview on related work dealing with models for representing the shape of objects. In Section 2.2 B-spline curves and surfaces are described. Methods that meet the requirements for fitting to point-clouds both in 2D and 3D are developed. Especially B-spline curves are investigated for approximating the boundaries of segments in image space and to be used for trimming the surfaces within their parametric domain. Section 2.3 describes how the planar regions of the range image are pre-segmented by clustering local normals. Using B-spline models of first and second order (four and nine control points) these planar patches are merged to continuous regions. Section 2.4 takes advantage of high level features given by parametric models. A graph-cut algorithm is applied for grouping those surface patches to meaningful objects. Section 2.5 describes how the surfaces of multiple views are merged together to form a complete model using the tracking system of Chapter 3. Results of each step are shown in Section 2.6 followed by a discussion of the introduced methods in Section 2.7. Figure 2.2 gives a high-level overview of our segmentation and reconstruction algorithm.

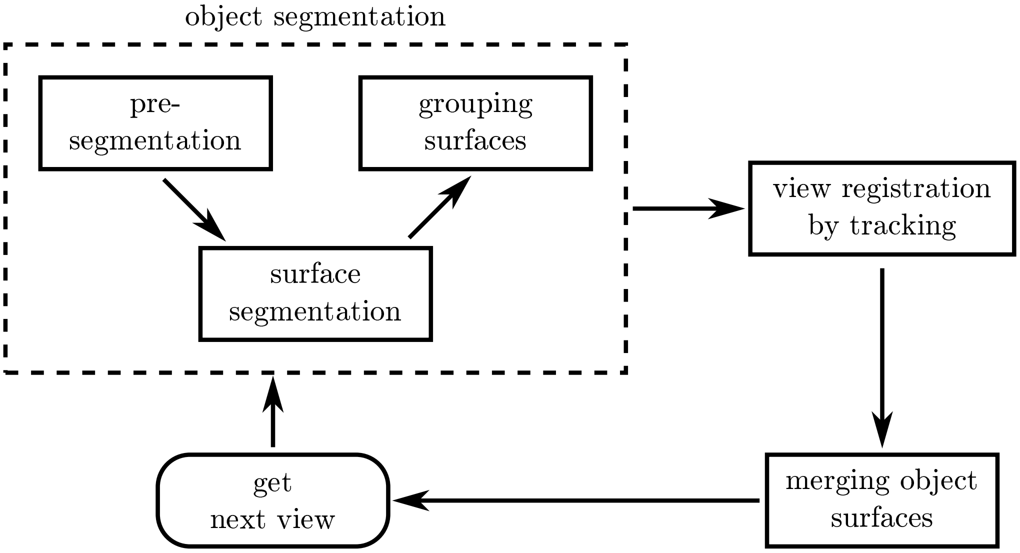


Figure 2.2: Starting from a single RGB-D image the surfaces and objects are segmented in a bottom-up approach. Object movement in front of the camera is visually tracked and new surfaces popping out from the object segmentation are merged with the existing shape model.

2.1 Related work

The importance of segmenting range images using geometric primitives has early been recognized by Besl et al. [4], where smooth graph surfaces are fitted to range data by least-squares minimization. Leonardis et al. [54] models data segments using bivariate polynomials, determining their order in a model selection scheme. In

[53] and [39] volumetric models, namely superquadrics, are introduced for simultaneous classification of image elements (segmentation) and estimation of the model parameters (reconstruction).

A popular method to represent objects for robotic grasping is to use shape primitives such as boxes, cylinders, spheres and cones [69] or triangulated meshes [87]. [10] and [5] use superquadrics as well, allowing for a higher variability and generalisation. Further [58] use continuous surfaces to compute the geodesic distance to deal with objects with holes (e.g. a mug with handle) whereas [21] use implicit surfaces for shape estimation and grasping.

The most simple geometric description of point-cloud segments are planes [37], followed by simple primitives like cylinders and spheres [38]. [101] use variational surface approximation, whereas [40] employ superquadrics. [14] developed a volumetric method to build complex models from range data.

[35] takes a mixture of five surfaces for modelling namely planes, conic surfaces, B-splines with four and nine control points and cluttered surfaces to represent natural scenes like bushes and trees. A jump-diffusion method allows them to segment range data efficiently within a Bayesian framework. Unfortunately the segments are not fused with the image data given and the range data is not corrected. Further a direct mapping from 2D image space into 3D world coordinates is not always accurate since their definition of B-spline surfaces does not ensure that every point of the image can be projected onto the 3D surface.

We abstract RGB-D data with plane and B-spline models to represent segments as well as their boundaries. Compared to other approaches our definition of surfaces provides a direct mapping from image into 3D world space. This allows to correct wrong and undefined values in the point-cloud leading to consistent depth and colour information. Adjusting the boundaries of the patches to the colour edges shows a considerable improvement of the segmentation. Further we present an algorithm to merge partial shape models from different views where we register the object poses using our tracking system with the TSD. Overlapping surfaces are merged by selecting the best fitting surface model. The boundaries of the resulting surfaces are found by employing our novel curve fitting algorithm, which is able to handle the unorganised data points in the parameter space of the surfaces. This leads to a shape model of low complexity and optimal degree of freedom.

2.2 B-spline fitting

A common way to model free-form curves and surfaces in computer-aided design (CAD), computer graphics and computer vision are B-splines and their generalisation the so-called *Non-Uniform Rational B-Splines* (NURBS). The reason for their popularity is the ability to represent all conic sections, i.e. circles, cylinders, ellipsoids, spheres and so forth. They are convenient to manipulate and possess useful mathematical properties, such as refinement through knot insertion, C^{p-1}-continuity for p-th order curves and the convex hull properties which we will exploit in Section 2.2.2.

2.2.1 Definition of B-spline curves and surfaces

A complete description of B-splines and their mathematical formulation would go far beyond the scope of this work, so we want to point the interested reader to the fundamental book [82]. A good overview on B-splines and NURBS is given in [12] which we want to summarize in this section. Let us start with the so called *knot vector* $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, a non-decreasing set of coordinates in the parameter space $\Omega_c = [\xi_1, \xi_{n+p+1}] \subset \mathbb{R}$, where n is the number of basis functions used for the B-spline curve and p is the polynomial order. The knots partition the parameter space into elements. With the knot vector in hand, the *B-spline basis functions* are defined with the *Cox-de Boor* recursion formula [13, 16]:

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

for $p = 0$ and for $p = 1, 2, 3, \ldots$, they are defined by

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.2}$$

There are several important features that come with this definition. The basis constitutes a partition of unity, that is, $\forall \xi$,

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1. \tag{2.3}$$

Also the basis function is point-wise non-negative over the entire domain, that is, $N_{i,p}(\xi) \geq 0$, $\forall \xi$, which becomes important in Section 2.2.2. Another important feature is, that each p-th order function has $p - 1$ continuous derivatives across the element. Figure 2.3a shows basis functions of order $p = 2$ with interpolating knots at $\xi = 0$ and $\xi = 5$.

B-spline curves

A B-spline curve $\mathbf{c}(\xi) : \Omega_c \to \mathbb{R}^3$ with the parametric domain $\Omega_c \subset \mathbb{R}$ is constructed by linear combinations of B-spline basis functions. Given n basis functions $N_{i,p}$ with $i = 1, 2, \ldots, n$, the n vector-valued coefficients of the basis functions, called *control vector* $\{\mathbf{b}_i \in \mathbb{R}^3\}$, defines the curve as

$$\mathbf{c}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \mathbf{b}_i. \tag{2.4}$$

The idea is to manipulate the B-spline curve $\mathbf{c}(\xi)$, by changing the values of the control points $\mathbf{b}_i$. The i-th control point defines the B-spline curve at its region of

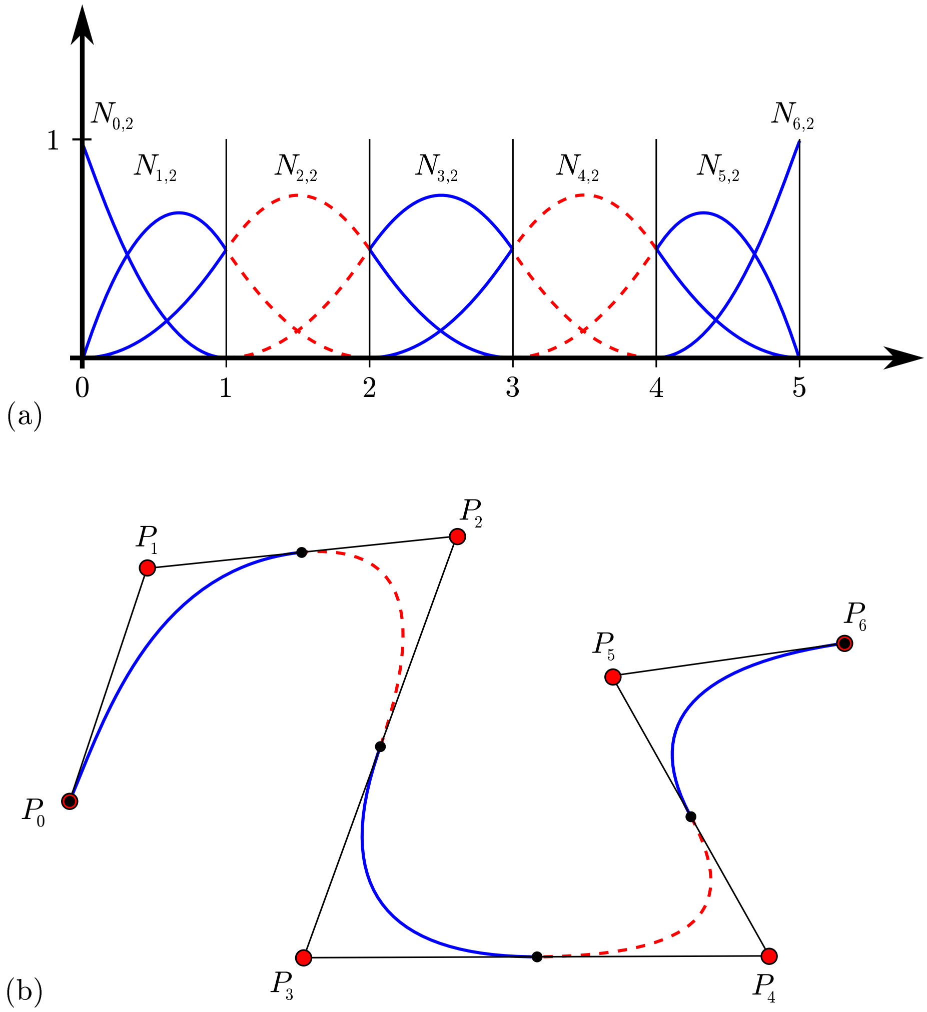


Figure 2.3: Figure 3.3 from [82]. (a) Quadratic B-spline basis functions with knot vector $\Xi = \{0, 0, 0, 1, 2, 3, 4, 5, 5, 5\}$. (b) A quadratic curve using the basis functions of (a).

influence determined by the basis function $N_{i,p}(\xi)$. A quadratic ($p = 2$) B-spline curve is shown in Figure 2.3b. An important characteristic of B-spline curves is that affine transformations of the curve are obtained by applying it directly to the control points. Furthermore B-spline curves obey a strong convex hull property resulting from the non-negativity and partition of unity properties of the basis, combined with the compact support of the functions. This leads to the fact that the B-spline curve is completely contained within the convex hull defined by its control points. This feature will be exploited for defining B-spline surfaces described in Section 2.2.2.

B-spline surfaces

A B-spline surface $\mathbf{S}(\xi) : \Omega_s \to \mathbb{R}^3$ with the parametric domain $(u, v) \in \Omega_s \subset \mathbb{R}^2$ is constructed by linear combinations of the tensor product of B-spline basis functions.

Given n, m basis functions $N_{i,p}$ and $M_{j,q}$ with $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$, the vector-valued coefficients, called *control grid* $\{\mathbf{B}_{i,j} \in \mathbb{R}^3\}$, defines the surface as

$$\mathbf{S}(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(u) M_{j,p}(v) \mathbf{B}_{i,j} \tag{2.5}$$

The same characteristics as for B-spline curves apply for B-spline surfaces. The derivatives in a given parametric direction may be determined from the respective one-dimensional basis function.

Trimmed B-spline surfaces

Due to the tensor product leading to an orthogonal parametric domain, the boundaries are four-sided with the properties given by the respective basis function of the surface. This is not always desirable (e.g. the cap of a cylinder is of 1^{st} polynomial order whereas its boundary may be represented by a quadratic NURBS curve). One possibility to get rid of the four-sided shape of a B-spline surface is to trim away areas that lie outside a certain region. The remaining part $\Omega_t \subset \Omega_s$ is defined on the parametric domain using closed B-spline curves. Figure 2.4 shows a sparse point-cloud of the Stanford bunny which is fitted and trimmed using B-spline curves and surfaces.[1]

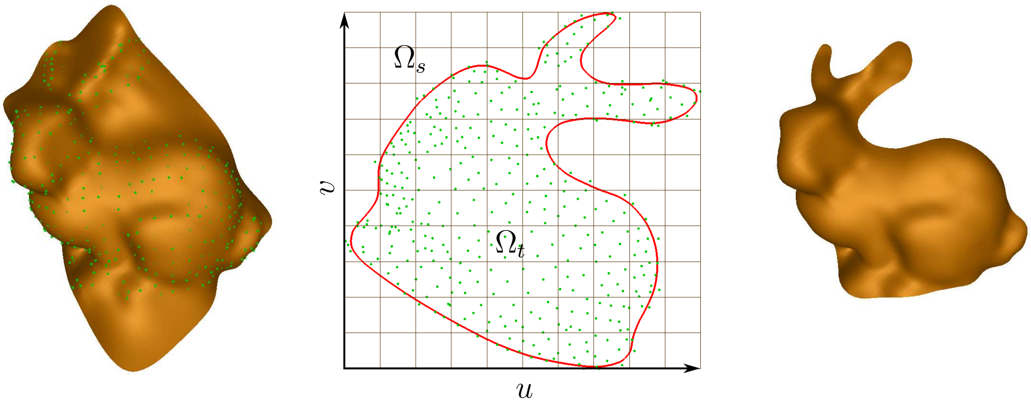


Figure 2.4: Trimming a B-spline surface (brown) using a B-spline curve (red). Left: The surface fitted to the data points (green). Middle: The curve modelling the outer contour defines a region Ω_t within Ω_s (grid). Right: The trimmed B-spline surface.

2.2.2 Fitting curves to 2D point-clouds

Fitting a B-spline curve of a certain degree of freedom to a set of points $\mathbf{p}$ is the task of manipulating the control points such that the distance between the points and the curve is minimized (Figure 2.5). This distance is usually determined by Newton's method.

---

[1] Video: `http://users.acin.tuwien.ac.at/tmoerwald/?site=4`

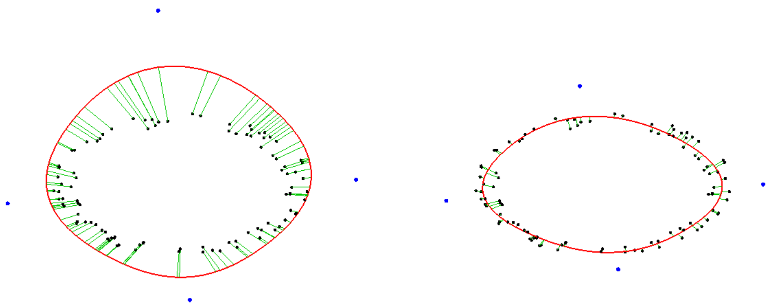


Figure 2.5: Curve-fitting: The distance (green) between the points (black) and the closed B-spline curve (red) is minimized by manipulating the control points (blue).

We aim to automatically find the outer boundaries of unorganised 2D point-sets with noise and clutter (Figure 2.4) and without any prior information of the complexity of the shape as shown in Figure 2.9. Further we do not use any initialization scheme to estimate the number of control points needed. We want to extend existing curve fitting techniques to make them more robust against clutter, noise and discontinuities. Hence, we require the curve fitting algorithm to cope with

1. unorganised point-clouds,
2. no explicitly given boundary points,
3. noisy data and clutter inside the boundary,
4. unknown required degrees of freedom (DOF),
5. deep and narrow concavities.

To solve these problems we have developed the following measures and methods:

- *Asymmetric Distance (AD)*, the distance measure for the minimization method to efficiently handle data points inside the boundary (1., 2., 3.).
- *Error-adaptive knot insertion (EAKI)*, which iteratively increases the DOF of the curve at regions of high error by adding control points (knots). This leads to a locally optimal resolution and allows for a trivial initialization (4., 5.).
- *Closest Point Strapping (CPS)*, a minimization constraint to force the curve towards inside points formed by deep and narrow concavities (5.).

Distance minimization

We assume the set $\{\mathbf{p}_k\}$ to consist of unorganised, scattered data points with considerable non-uniformly distributed noise and heavy clutter. A commonly used formulation of fitting a B-spline curve to a set of points $\{\mathbf{p}_k\}$ is to minimize the objective function

$$f = \frac{1}{2}\sum_{k=1}^{n} e_k + w_s f_s \tag{2.6}$$

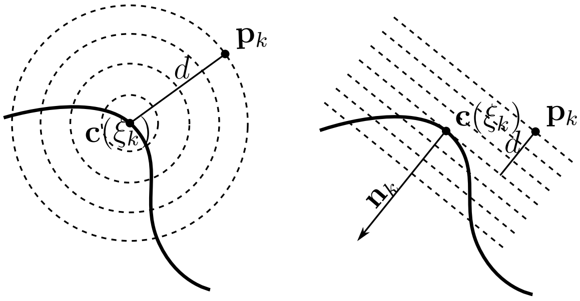
with respect to the control vector $\mathbf{b}$, where e_k is an error term describing the distance between the data points and the curve. A simple and commonly used distance is the *Point Distance* (PD) defined as

$$e_{PD,k} = ||\mathbf{c}(\xi_k) - \mathbf{p}_k||^2 \tag{2.7}$$

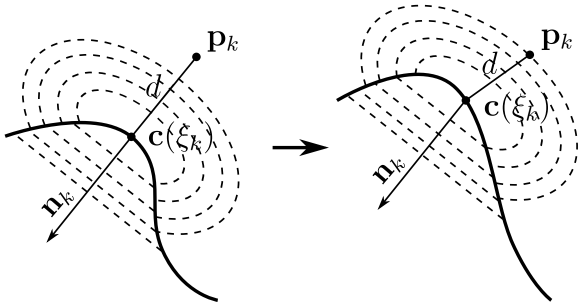
Typically a weighted smoothing term (ST) $w_s f_s$ is used to obtain a visually satisfying solution.

$$f_s = \int_{\Omega} ||\mathbf{c}''(\xi)||^2 dt \tag{2.8}$$

The *footpoint* ξ_k, with $\mathbf{c}(\xi_k)$ being the closest point to $\mathbf{p}_k$, with respect to Euclidean distance, is evaluated by using Newton's method.



(a) point distance (b) tangent distance



(c) squared distance from one iteration to the next

Figure 2.6: Iso-value curves for point distance (PD), tangent distance (TD) and squared distance (SD). Note that for SD the normal vector $\mathbf{n}_k$ does not change during optimization.

Blake and Isard [6] introduced a *Tangent Distance* (TD) term to be minimized, leading to faster convergence.

$$e_{TD,k} = [(\mathbf{c}(\xi_k) - \mathbf{p}_k)^T \mathbf{n}_k]^2 \tag{2.9}$$

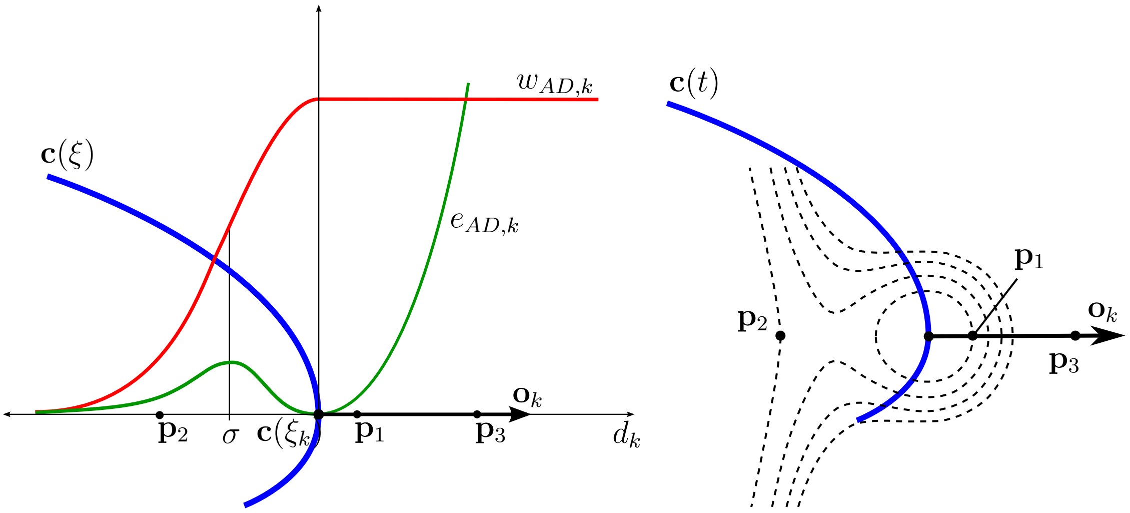
where $\mathbf{n}_k$ is the normal vector of the curve at the point $\mathbf{c}(\xi_k)$. The drawback of the TD, as shown in [99], is that the method is less robust with respect to local minima.

Therefore Wang et al. [99] introduced the *Squared Distance* (SD) term to benefit from both, the robustness of PD and fast convergence of TD.

$$e_{SD,k} = \begin{cases} \frac{d}{d-\rho}\left[(\mathbf{c}(\xi_k)-\mathbf{p}_k)^T\xi_k\right]^2 + \left[(\mathbf{c}(\xi_k)-\mathbf{p}_k)^T\mathbf{n}_k\right]^2 & \text{if } d < 0 \\ \left[(\mathbf{c}(\xi_k)-\mathbf{p}_k)^T\mathbf{n}_k\right]^2 & \text{if } 0 \leq d < \rho \end{cases} \tag{2.10}$$

where $\mathbf{t}_k$ is the tangent vector at the curve point $\mathbf{c}(\xi_k)$. Let ρ be the curvature of the curve and d be the signed distance, where $d < 0$ if $\mathbf{p}_k$ is on the opposite side of $\mathbf{n}_k$ and $d \geq 0$ if they are on the same side. Note that $\mathbf{n}_k$, $\mathbf{t}_k$ and ρ do not change during iteration. Figure 2.6 shows the three distance measures PD, TD and SD. For a more detailed discussion let us refer to the paper of Wang et al. [99]. We have applied our approach using $e_{PD,k}$, $e_{TD,k}$ and $e_{SD,k}$ experiencing similar behaviour as described in the respective literature.

Asymmetric distance (AD)



(a) Asymmetric distance function for PD (b) Iso-value curves of $e_{AD,k}$ for PD

Figure 2.7: *Asymmetric Distance:* Weighting function $w_{AD,k}$ (red) and asymmetric distance (AD) term $e_{AD,k}$ (green) for fitting the point $\mathbf{p}_k$ attached to the footpoint of the B-spline curve $\mathbf{c}(\xi)$ (blue). Two opposing points $\mathbf{p}_1$ and $\mathbf{p}_2$ lie on the same iso-value curve even when the Euclidean distance of $\mathbf{p}_2$ is higher. In other words, a point $\mathbf{p}_3$ outside the curve, with the same Euclidean distance from $\mathbf{c}(\xi_k)$ as a point $\mathbf{p}_2$ inside, causes a much higher error. Note that in general the outward normal vector $\mathbf{o}_k$ does not point in the same direction as the second derivative of the curve $\mathbf{c}''(\xi_k)$.

As depicted in Figure 2.15 segmentations are often subject to heavy clutter and outliers at the boundary as well as inside. Figure 2.4 shows an example where the point-cloud is not organized and a boundary other than the convex hull can not be defined properly. This leads to the idea of an asymmetric distance (AD) term such that points that are inside the boundary are weighted less than points outside. Let $\tilde{\mathbf{p}}_k$

be the vector pointing from $\mathbf{c}(\xi_k)$ to $\mathbf{p}_k$ and $\mathbf{o}_k$ be the outward pointing unit normal vector of the curve at ξ_k. We define the asymmetric weighting function $w_{AD,k}$ as

$$w_{AD,k} = \begin{cases} e^{-\frac{d_a^2}{2\sigma^2}} & \text{if} \quad d_a < 0 \\ 1 & \text{if} \quad d_a \geq 0 \end{cases} \tag{2.11}$$

where σ defines the width of the transition of the weighting function with respect to the signed distance

$$d_a = \tilde{\mathbf{p}}_k^T \mathbf{o}_k \tag{2.12}$$

The weighting function is multiplied by any distance term e_k, inducing our new asymmetric distance term illustrated in Figure 2.7.

$$f_{ad} = \frac{1}{2} \sum_{k=1}^{n} e_{AD,k} \tag{2.13}$$

$$e_{AD,k} = \begin{cases} w_{AD,k} e_{PD,k} \\ w_{AD,k} e_{TD,k} \\ w_{AD,k} e_{SD,k} \end{cases}$$

This forces the curve to the outer boundary points, and strongly de-weights points inside, which means that also points of concavities are not considered immediately. Fortunately the half bell-shaped function $w_{AD,k}$ iteratively closes the gap between the curve and the data points. This is different to most of the other approaches (i.e. [82, 6, 99]) where all points are treated the same in a global sense. For the remainder of this paper we define $e_k = e_{AD,k} = w_{AD,k} e_{SD,k}$ if not stated otherwise. Figure 2.8 shows how easy it is to confuse minimization methods using PD, TD or SD without our AD term.

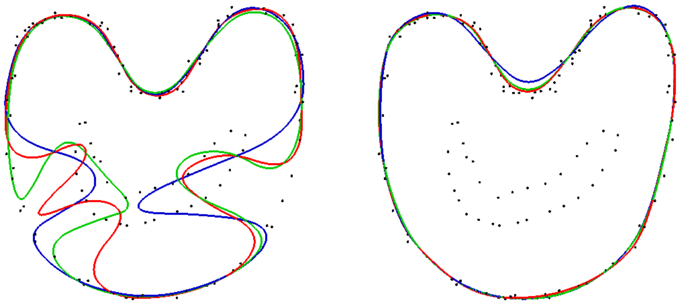


Figure 2.8: Typical problem when adding some clutter inside the boundary of a dataset such as Figure 10 of [99] (left). Solving the problem using the asymmetric distance (AD) of our approach (right). Red: point distance [82]. Green: tangent distance [6]. Blue: squared distance [99].

Error-adaptive knot insertion (EAKI)

In real world applications the degrees of freedom of the boundary is usually unknown. Initialization is typically done by user input or by some estimation scheme as in [99, 103, 80]. In the work of [18] knots are inserted or removed depending on the distance between neighbouring knots. In our opinion this is quite counter-intuitive, since segments where a low resolution of the curve already fits large parts of the contour do not require refinement. We want to introduce a new method that automatically adapts the DOF by iteratively inserting knots to the B-spline curve at points where the error is above the accuracy specified by the user. This leads to a non-uniformly distributed knot vector, with a high resolution at regions of high curvature and a low resolution at straight segments. In other words, knots are placed where they are needed, given a certain accuracy (see Figure 2.9).

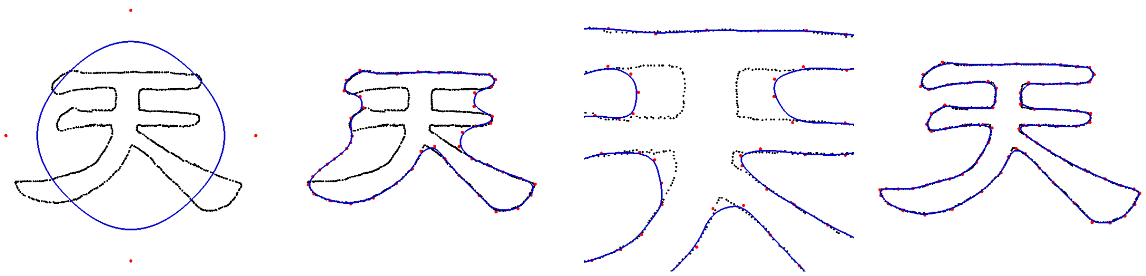


Figure 2.9: *EAKI:* Control points (red) are iteratively inserted, automatically adapting the curve (blue) to the required degrees of freedom of the outline of the Chinese character *tian*. From left to right: Initial curve, 10, 15 and 30 iterations with 4, 61, 73 and 82 control points respectively. Note the simple initialization and the iterative increase of the DOF by knot insertion, while the AD term fastens the curve to the data points.

During each fitting iteration we measure the point distance $e_{PD,l}$ from every curve point $\mathbf{c}(\xi_l)$ to the closest point $\mathbf{p}_l$ of the point cloud, where ξ_l are the midpoints of two adjacent elements of the knot vector. If the distance exceeds the accuracy specified by ε_a, a new knot is inserted at the curve parameter point ξ_l.

Closest Point Strapping (CPS)

Unfortunately noise at the boundary and sharp turns may stop the effect of iteratively closing the boundary of the AD term since it de-weights data points inside the boundary (Figure 2.10). Increasing σ might work for some cases, but causes problems when neighbouring boundaries are close to each other.

This leads to the idea of explicitly finding data points orthogonal to the boundary and strap the curve to these points. Starting form Equation (2.6), we add the term

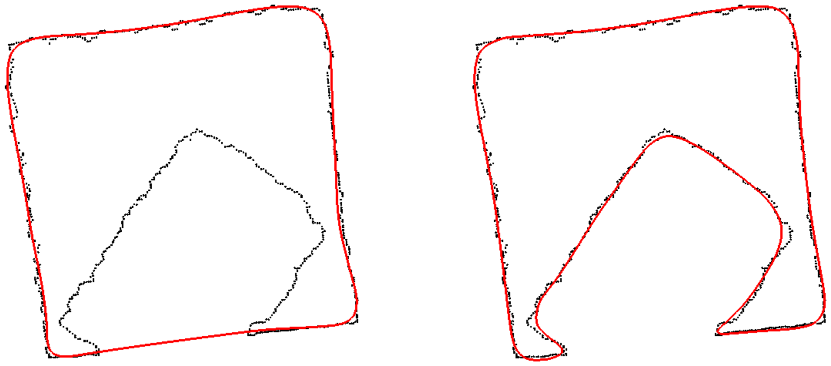


Figure 2.10: *Sharp turns:* Without concavity filling methods (left). With *Closest Point Strapping* (right).

$w_c f_c$ to the functional to be minimized,

$$\begin{aligned} f &= f_{ad} + w_s f_s + w_c f_c \\ f_c &= \textstyle\sum_l ||\mathbf{c}(\xi_l) - \mathbf{p}_l||^2 \end{aligned} \tag{2.14}$$

where $\mathbf{p}_l$ is the closest point to the B-spline curve at ξ_l. In a similar manner as for EAKI, we find the closest point for each midpoint ξ_l of two neighbouring elements of the knot vector. The difference is that we are not using the Euclidean distance to find $\mathbf{p}_l$, since we are searching for points within the curve and close to the straight line normal to the curve. Therefore we define the distance as

$$d_o = \begin{cases} 0 & \text{if} \quad |\tilde{\mathbf{p}}_l| = 0 \\ \infty, & \text{if} \quad \mathbf{o}_l^T \tilde{\mathbf{p}}_l \geq 0, \, |\tilde{\mathbf{p}}_l| \neq 0 \\ \frac{\tilde{\mathbf{p}}_l^T \tilde{\mathbf{p}}_l}{|\mathbf{o}_l^T \tilde{\mathbf{p}}_l|}, & \text{if} \quad \mathbf{o}_l^T \tilde{\mathbf{p}}_l < 0, \, |\tilde{\mathbf{p}}_l| \neq 0 \end{cases} \tag{2.15}$$

which results in iso-value curves as shown in in Figure 2.11. As we are minimizing Equation (2.14) the curve is strapped to data points $\mathbf{p}_l$ behind the sharp turn. Once the curve is close enough ($d_o < \sigma$) to points in the neighbourhood of $\mathbf{p}_l$ the closing effect induced by AD continues.

Implementation

We have implemented the concepts above using *openNURBS* [67]. All of the software is available within the *Point Cloud Library* [88][2]. An outline of our approach is shown in Algorithm 1.

---

[2] `http://www.pointclouds.org`

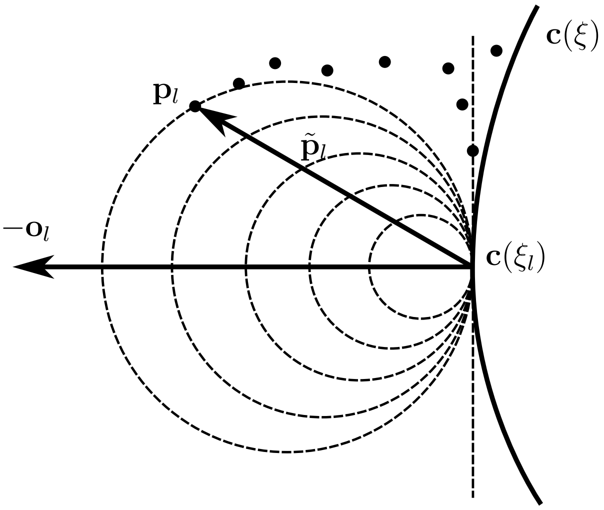


Figure 2.11: *Closest Point Strapping:* Iso-value curves for finding the closest point to the curve at point $\mathbf{c}(\xi_l)$, with respect to the outward pointing normal vector $\mathbf{o}_l$. Note that with the distance metric d_o the point $\mathbf{p}_l$ is closer than the point right above $\mathbf{c}(\xi_l)$ and therefore used for fitting the curve at ξ_l.

Initialization: For initialization we simply calculate the bounding circle of the point cloud and set the 4 initial control points of the closed periodic B-spline curve to lie on this circle while being shifted by $\pi/4$ from each other (see Figure 2.9 left).

Distance minimization: For minimization of the functional in Equation (2.14) we assemble the three terms into a single linear system. Considering Equation (2.4) it is easy to rewrite the first term as

$$\mathbf{f}^k = \mathbf{K}^k \mathbf{b} \tag{2.16}$$

where $\mathbf{f}^k$ is called *force vector* and contains all the points $\mathbf{p}_k$ of the point-cloud to be fitted. The *stiffness matrix* $\mathbf{K}^k$ contains the basis functions at the respective footpoints ξ_k in matrix form and $\mathbf{b}$ is the vector of control points. The linearisation of the third term of Equation (2.14) responsible for CPS is analogue to the first term.

$$\mathbf{f}^l = \mathbf{K}^l \mathbf{b} \tag{2.17}$$

where $\mathbf{f}^l$ contains the closest points $\mathbf{p}_l$ and $\mathbf{K}^l$ the basis functions at the footpoints ξ_l in matrix form. The smoothing term of Equation (2.8), minimizing the second derivatives of the functional, is approximated by a regularisation matrix $\mathbf{R}$, which is multiplied with the control vector $\mathbf{b}$. $\mathbf{R}$ is constructed such that the position of a certain control points tends to lie in the middle of its neighbours in the control grid.

$$\mathbf{0} = \mathbf{R}\mathbf{b} \tag{2.18}$$

Algorithm 1 Curve fitting algorithm

```
initialization
while $r < R$ or $f_d > \varepsilon_c$ and $||\mathbf{c}(\xi_l) - \mathbf{p}_l||^2 > \varepsilon_a$ do    ▷ Termination

    for all points $p_k$ do    ▷ parametrization
        find footpoint $\xi_k$ for $\mathbf{p}_k$ using Newton's method
    end for

    minimize $f = f_{ad} + w_s f_s + w_c f_c$    ▷ AD + ST + CPS

    for all midpoints $\xi_l$ of $\mathbf{c}$ do    ▷ EAKI
        find closest point $\mathbf{p}_l$ of $\mathbf{c}(\xi_l)$
        if $||\mathbf{c}(\xi_l) - \mathbf{p}_l||^2 < \varepsilon_a$ then
            insert new knot at $\xi_l$
        end if
    end for

end while
```

Assembling Equation (2.16), (2.18) and (2.17) into one single system yields

$$\begin{bmatrix} \mathbf{f}^k \\ w_s \\ w_c \mathbf{f}^l \end{bmatrix} = \begin{bmatrix} \mathbf{K}^k \\ w_s \mathbf{R} \\ w_c \mathbf{K}^l \end{bmatrix} \mathbf{b} \tag{2.19}$$

with w_s and w_c being the weights for smoothing and CPS. For solving this linear system efficiently we use the unsymmetric multifrontal sparse LU factorization package (UMFPACK) of [15].

Error-adaptive-knot-insertion: The implementation of EAKI is quite straight forward. Since the closest points at the footpoints ξ_l are anyway computed during closest point strapping, this distance only has to be compared to the threshold ε_a set by the user.

Termination: According to Algorithm 1 the optimization loop terminates if the following conditions are met.

- The number of iterations exceeds a maximum R specified by the user.
  OR
- The maximum distance of the curve at $\mathbf{c}(\xi_l)$ to the closest point $\mathbf{p}_l$ is below the threshold ε_a. This means that the curve is close enough to the data points.
  AND
- f_c is lower then a user specified threshold ε_c. This means that the curve fits all data points sufficiently.

Image based curve fitting

The RGB-D image given by a sensor such as the Microsoft Kinect typically suffers from heavy noise. Usually depth edges do not correspond to colour edges. Areas with large deviation in depth are often missing points or, if they are available, their values are completely wrong. To solve this problem we propose to fit a B-spline curve to the contour of surface segments resulting from the segmentation algorithm described in Section 2.3. Therefore we formulate the fitting problem such that the curve aligns to the edges given by the colour image.

For each surface patch of the segmentation we compute the contour of the points in image space shown in the top left of Figure 2.12. For this we parse all pixels of the segmented region and mark them if at least one of the 4-connected neighbourhood pixels does not belong to the region. The marked pixels include the outer boundary, but also inner clutter points if the region is not completely dense or has holes. Hence, to get the outer ordered boundary we trace the pixel chains with an 8-connected neighbourhood kernel and select the largest chain which surrounds the inner points. To get a more consistent and accurate representation of the contour we search for points in its neighbourhood that lie on colour edges. Each boundary pixels is substituted by the closest colour edge pixel (Figure 2.12 top-right). For calculating the edges of the colour image we use the Canny edge detection algorithm [7] on the grey-scale image. The resulting set of points are used for initialisation and fitting of the B-spline curve. The bottom right of Figure 2.12 shows the resulting curve.

2.2.3 Surface fitting

Similar as for curves, fitting a surface is the task of finding values for the control points such that the distance between the points and the surface is minimized. Again Newton's method is applied for finding the closest point on the surface. Figure 2.13 shows a surface of 3^{rd} order and 16 control points fitted to a set of points. The surface is initialised using the principal-component-analysis (PCA) where the plane is defined by the mean of the points and the two largest eigenvectors. In general for this kind of surface fitting, the choice of the parameter space is not important, in contrast to image based surface fitting.

Image based surface fitting

As shown on the left of Figure 2.14 we define the parametric domain $\Omega_s \subset \mathbb{R}^2$ with $(u, v) \in \Omega_s$, of the surface $\mathbf{S} : \Omega_s \rightarrow \mathbb{R}^3$ to be a subset of the image space of the camera. In more detail we define Ω_s to be the bounding box of the control points of the B-spline curve. Together with the convex hull property of B-splines this ensures that every point inside the curve lies inside the bounding box of the control points and with Equation (2.5) has its corresponding point in $\mathbb{R}^3$. This means that these points and also the curve itself can simply be transformed to $\mathbb{R}^3$ as shown on the right of Figure 2.14. For an organized 3D point-cloud this means that we can skip

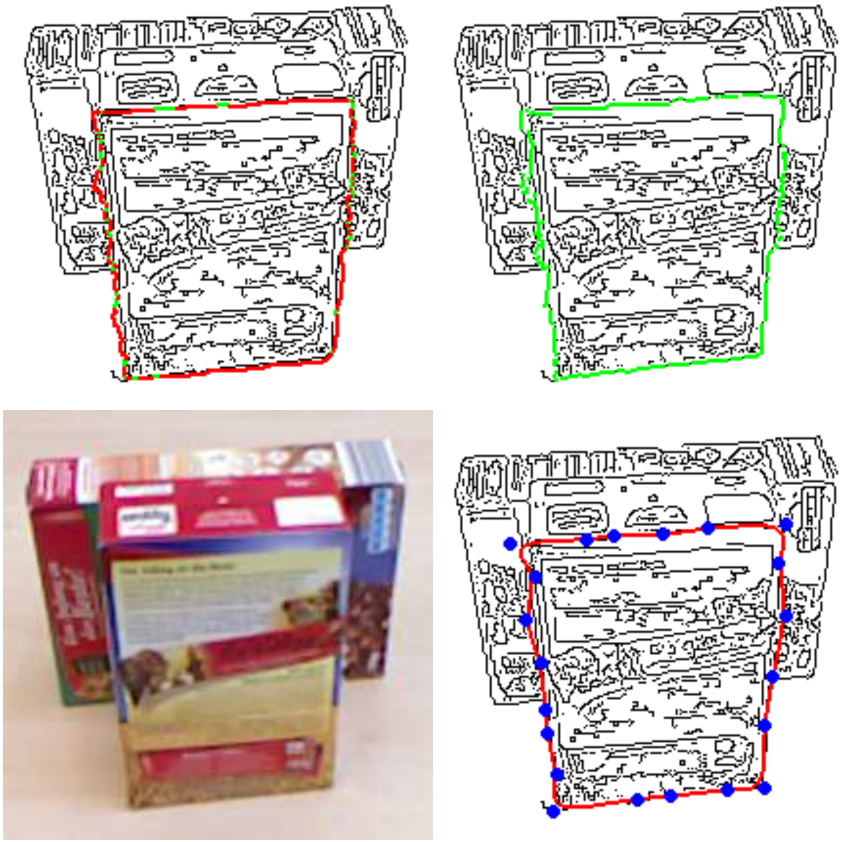


Figure 2.12: Adjusting the contour points of the surface segmentation of Section 2.3 to the edges of the image. Green points lie on image edges (black) in contrast to red points. Top-Left: Contour of the point-cloud patch. Top-Right: Contour updated with respect to the image edges. Bottom-Left: Colour image. Bottom-Right: Curve fitted to the updated contour.

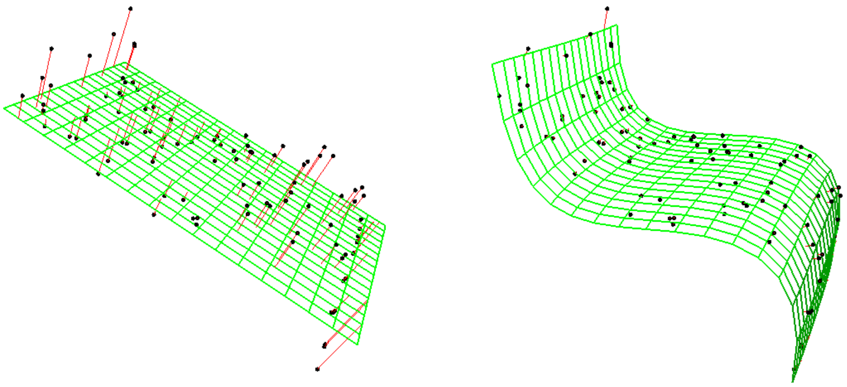


Figure 2.13: Fitting a B-spline surface (green) by minimizing the distance (red) between points (black) and the surface.

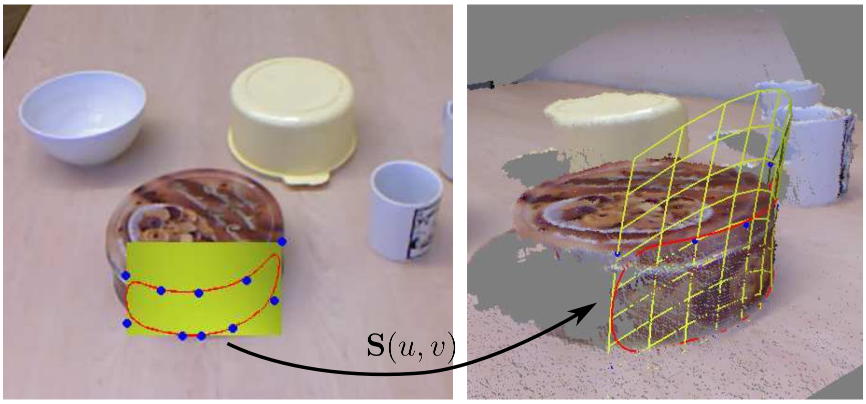


Figure 2.14: The parametric domain Ω_s (yellow) defined on the image space (left) is mapped into 3D world coordinates by the B-spline surface function $\mathbf{S}(u, v)$ (right). The control points (blue) of the curve (red) define the boundaries of Ω_s.

the expensive search for the closest points on the surface, since the pixel coordinates are equal to the footpoints (u, v) in the parametric domain of the surface.

Similar to B-spline curve fitting we again minimize the distance between the 3D-points and the B-spline surface. As we require the projection of the B-spline surface to remain unchanged we constrain the control points such that they only vary in z-direction. Since the mapping from image space into 3D coordinates using the pinhole camera model is not similar, in a geometric sense, the point distance minimization (Equation (2.7)) is not suitable. Hence we apply the tangent distance for fitting the B-spline surface.

$$e_{TD,k} = [(\mathbf{S}(u_k, v_k) - \mathbf{p}_k)^T \mathbf{n}_k]^2 \tag{2.20}$$

Equivalent to Equation (2.9) $\mathbf{S}(u_k, v_k)$ is the closest surface point to $\mathbf{p}_k$ and $\mathbf{n}_k$ is the surface normal at this very position.

Segmentation refinement and point-cloud completion

With this representation at hand, points that do not lie within the contour of a certain patch, are reassigned to the neighbouring segments. If no depth value is available, it is calculated by mapping the image point to the corresponding 3D surface. Computing the surfaces and curves for all segments leads to a highly accurate, continuous, consistent and efficient representation of the scene which significantly improves the end result as we will demonstrate in the Section 2.6.

2.3 Surface segmentation

The work presented in this section was developed together with my colleagues Andreas Richtsfeld and Johann Prankl and is described in detail in [83] and [73]. 3D cameras, such as Microsoft's Kinect or Asus' Xtion provide RGB-D data, consisting of a colour image and the associated depth information for each pixel. We aim for continuous regions (smooth surface patches) by fitting planes and B-splines of higher polynomial order to account for curved surfaces.

2.3.1 Pre-segmentation

The task of pre-segmentation is to cluster neighbouring pixels to planar patches without discontinuities using locally calculated normals. Therefore we exploit the relationship between 2D image space and associated depth information of organized point-clouds (i.e. range images).

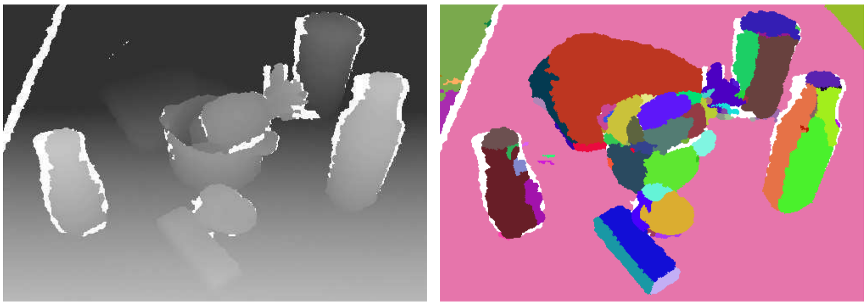


Figure 2.15: Segmentation of planar patches of an RGB-D image by clustering points with similar local normals.

Normal calculation

An easy way to calculate the normals of a point-cloud is to locally fit planes to neighbouring 3D points. We use organized point-clouds as obtained from the Microsoft Kinect, where a kernel defines the neighbourhood of a certain point.

Normal clustering

Points are clustered recursively with respect to the angular deviation α of their normal compared to already clustered patches. If α is bellow a certain threshold it is added to the patch it was compared to. The normal of the already clustered points is simply the mean of all point normals.

2.3.2 Model selection

In the last section continuous planar patches were extracted from RGB-D data. Parametrization of these patches to certain surface models allows for a more compact representation. For surface segmentation two parametric models are chosen, planes to represent simple planar patches and B-splines allowing to represent curved surfaces of higher polynomial order. Note that B-splines of polynomial order $p = 1$ could also represent planes and we could thus skip explicit plane models, but they are more efficient in terms of data size, fitting and processing (e.g. intersections, distance).

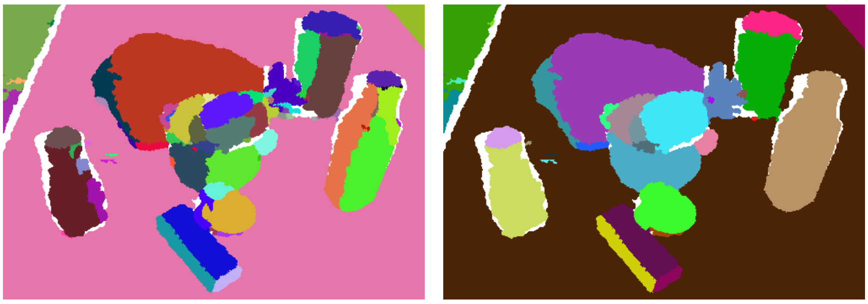


Figure 2.16: Merging planar regions using B-spline surfaces of higher polynomial order.

Neighbouring patches are merged after parametrization, if a joint parametric model fits better than the two individual models. To come to a decision, model selection with a minimum description length (MDL) criterion [54] is used. The sum of savings

$$S_h = \kappa_0 S_{area} - \kappa_1 S_{model} - \kappa_2 S_{error} \tag{2.21}$$

for models of neighbouring patches are compared with savings of a model fitted to a merged patch and in case the latter is larger the individual patches are substituted. S_{area} corresponds to the number of points that can be explained by a hypothesis h. S_{error} represents the costs for describing the error made by h. The weights κ_0, κ_1 and κ_2 can be determined on an information-theoretical basis, or they are adjusted for preferring a certain term. S_{model} encodes the complexity of the models used. In our case we use the number of control points n_{cps} defining the surface and normalize it over all surfaces involved. Given that we want to choose among m primitives the model complexity of the i-th model is given by

$$S^i_{model} = \frac{1}{\sum_{j=1}^{m} n^j_{cps}} n^i_{cps} \tag{2.22}$$

We also use the MDL for the reconstruction of the object shape described in Section 2.5. Table 2.1 lists some geometric primitives and the number of control points defining the surface. Further we allow for refinement by knot insertion increasing the number of control points. For more details on the MDL formulation we want to refer to the work of [52].

Table 2.1: Geometric primitives

Primitive	# control points
Plane	3
Quadratic B-spline patch	9
B-Spline Cylinder	16
Quadratic B-spline cylinder	24
B-spline sphere	26

2.4 Object segmentation

The work on object segmentation was established together with my colleague Andreas Richtsfeld and is explained in more detail in [83]. In the previous section we explained how to find a representation of a point cloud consisting of geometric primitives such as planes and B-spline surfaces. The main idea was to segment with respect to continuous regions using range images. Now we are interested in clustering these regions into object hypothesis. In contrast to Section 2.3 we are using colour information as well, additionally to depth (Figure 2.17).

2.4.1 Grouping of parametric surfaces

Starting form the segmentation of Section 2.3, we try to find relations between surfaces that indicate if they belong together. Inspired by the principles of Gestalt [100] features like curvature, normals direction, spatial location, colour and texture are employed to define these relations. A support-vector-machine (SVM) is trained from hand-annotated data, which maps the set of relations between two surfaces to a single probability value. We can now build a graph, where the surface patches are nodes and the relations are the edges. This allows us to use graph-cut to segment objects in a global sense. The lower-right of Figure 2.17 shows the result of object segmentation on the example already used in Section 2.3.

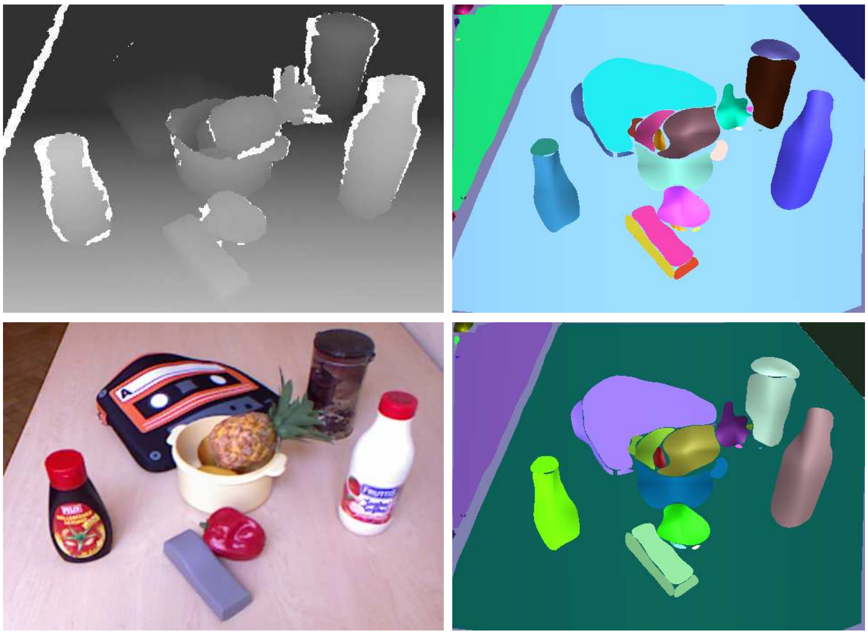


Figure 2.17: Grouping parametric surfaces to objects using depth and colour information. Left: Depth and colour image. Right: Surface- and object segmentation.

2.5 Learning

Given the object segmentation of a single view as in Figure 2.17, we now aim to reconstruct the shape of the object using multiple points of view. We consider the cognitive robotic scenario, where the representation of the shape requires to be extendible. This is different from typical reconstruction approaches [30, 31, 98, 104, 57, 79], which are taking advantage of global optimization techniques. Of course these could be applied to our approach after several key-frames have been recorded, leading to more accurate models. However, since this is not in the sense of an extendible representation we do not discuss global optimization for multi-view reconstruction in this work.

2.5.1 Object registration

Starting from the segmentation of a single view, the object of interest is selected by a tutor. Colour information is mapped onto the shape geometry by simply projecting it into image space. Then the tracking system described in the subsequent Chapter 3 is employed to follow the pose of the object in real-time while moving it in front of the camera sensor. Once another good view point is indicated (see Section 3.3) a key-frame is captured containing depth- and colour information. The object segmentation algorithm is performed on this key-frame.

2.5.2 Surface merging

The surfaces of the object in the new key-frame are merged with the current surfaces. Simply adding the new surfaces to the object will lead to overlaps, which need to be resolved accordingly. Therefore we first need to identify the segmented object in the new view as the object with the largest area of overlap, with respect to the projection of the tracked appearance model. Afterwards surfaces which significantly overlap are merged, resulting in a single surface patch with the proper model.

Overlap evaluation

For all surface patches of the current object we compute the overlap with all the surface patches of the new object in image space. Therefore we project both surfaces into 2D image space and there consider their intersection. For each point being part of this intersection we compute the angular deviation of the surface normals in world coordinates. The sum of the angular deviation over the region of intersection defines our measure of overlap. This overlap is then normalized by the number of points of the two surfaces in image space yielding two values. If one of the normalized overlap values exceeds a certain threshold the surface pair is merged. Figure 2.18 shows an example where two quadratic B-spline patches are merged (top row) and one where the normalized overlap value is too low for merging (bottom row).

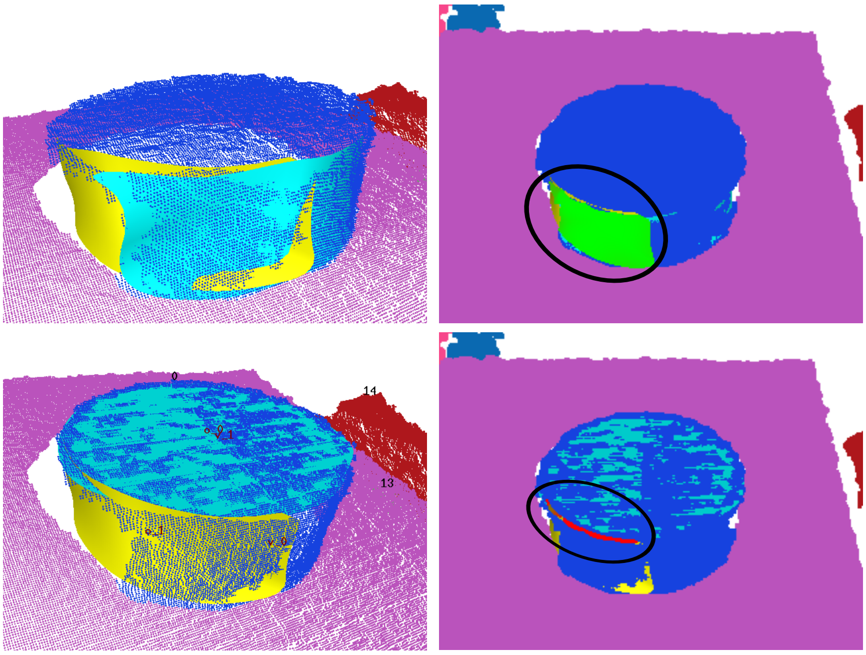


Figure 2.18: Overlap of a surface of the existing model (yellow) with two new surfaces (cyan). The overlap in image space (right) is evaluated taking into account the surface normals. Green indicates high- and red low overlap w.r.t. deviation of normals. As a result the surfaces in the top row are merged in contrast to the bottom row.

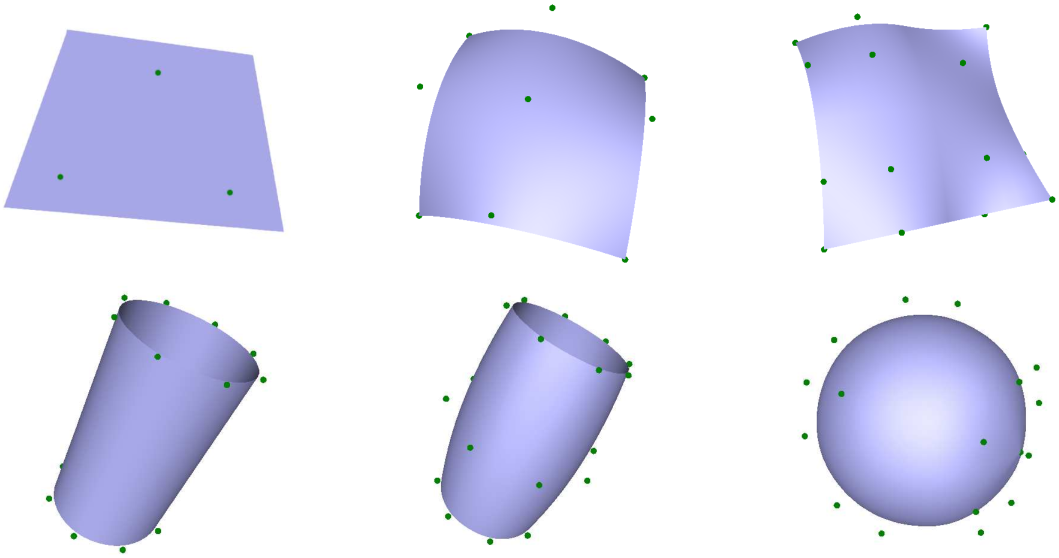


Figure 2.19: Geometric primitives used for fitting point-clouds in $\mathbb{R}^3$. The best model according to Equation (2.21) is selected. From top-left to bottom-right: Plane, quadratic B-spline patch, refined quadratic B-spline patch, B-spline cylinder, quadratic B-spline cylinder, B-spline sphere.

Model selection

Consider that the 3D-points from previous views belonging to a certain surface are given. Then the union of these points and the points of the surface of the current view form the data we want to fit with a geometric model for merging. So in this sense we substitute the two surfaces by a single one rather than merging them. We select the best fitting surface by comparing their minimum description length according to Equation (2.21). Figure 2.19 shows the surface primitives used for merging. To define a boundary for the resulting surface we employ the curve fitting algorithm of Section 2.2.2 within its parametric domain. Figure 2.20 shows how quadratic B-spline surfaces are merged by substitution.

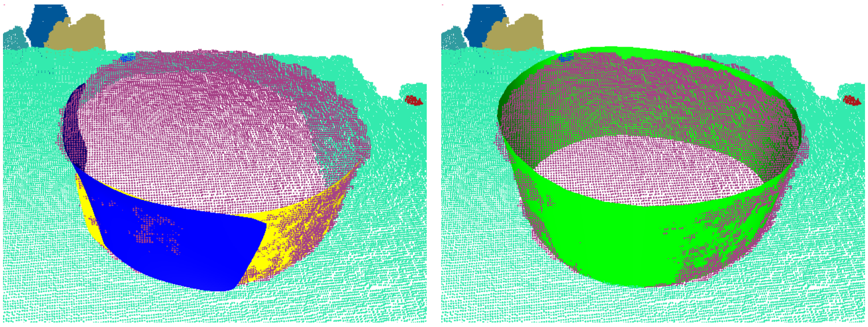


Figure 2.20: Two quadratic B-Spline patches are substituted by a B-spline cylinder.

2.6 Results

2.6.1 B-spline curve fitting

We employ our method in a generic way to real world data for finding the boundary of point-clouds on manifolds like B-spline surfaces for reconstruction. Please note, that segmentation and surface fitting is not the focus of this Subsection and is therefore treated as given.

Continuous region segmentation: Modern methods for image segmentation often take advantage of depth information such as [85, 83] who try to identify continuous regions using RGB-D information from sensors like the Microsoft Kinect as illustrated in Figure 2.21. Please note the complex shape of the right side of the desk, the deep concavity of the segment in the foreground, as well as the heavy clutter occurring at the right side[3]. The noise is estimated to adjust accuracy ε_a for EAKI and transition width σ for AD. ($\sigma = 0.01$, $w_s = 1.0$, $w_c = 1.0$, $\gamma = -1.0$, $\varepsilon_d = 0.5$, $\varepsilon_a = 0.01$)

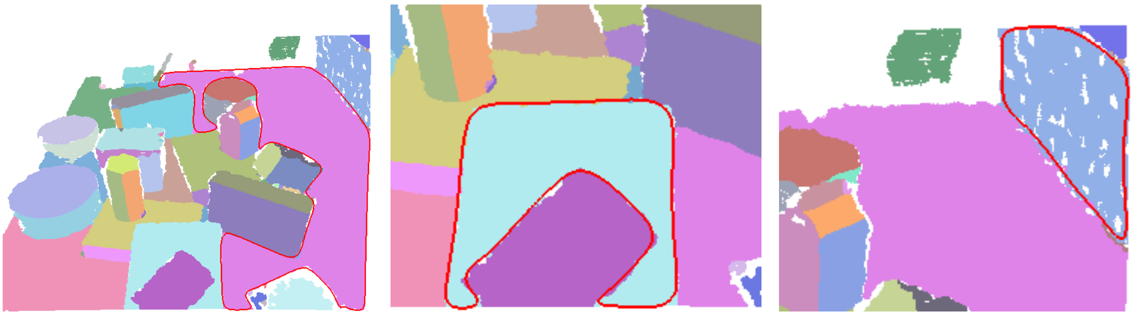


Figure 2.21: Results of our approach when finding the contours of image segments with complex shapes, deep concavities and clutter inside the region.

Comparison to PD, TD and SD: A qualitative comparison of methods like PD, TD and SD with our method is quite hard, since the former are not designed for treating clutter inside the boundary and rely on proper initialization schemes to obtain good solutions. However, as we want to point out the significance of our approach in real world scenarios we show cases where PD, TD and SD is not appropriate in contrast to AD. Figure 2.22 shows how a single outlier forces the curve not to lie on the boundary.

Due to the EAKI our approach automatically determines the required DOF by inserting knots at points where the error is above the accuracy required. This allows for a simple and fast initialization. Figure 2.23 shows that SD only converges to a satisfying solution if a sufficiently good initialization is given. (i.e. placement of enough control points close to the boundary). On the other hand AD leads to a satisfying solution with a very simple initialization starting with only 4 control

---

[3] Video: `http://users.acin.tuwien.ac.at/tmoerwald/?site=4`

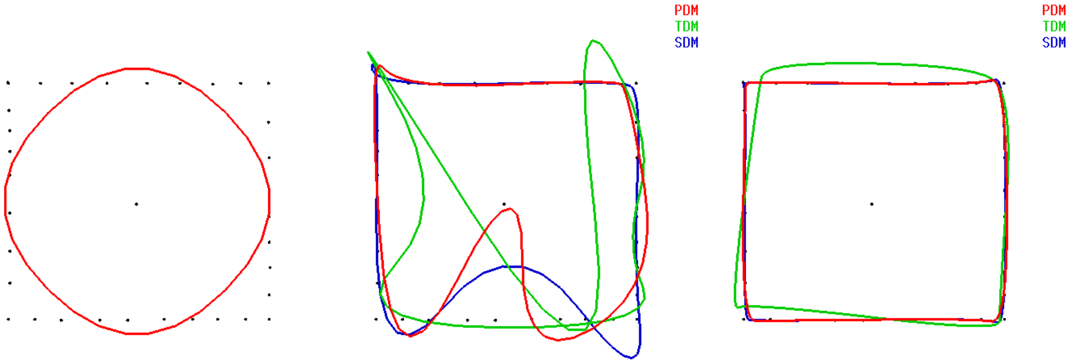
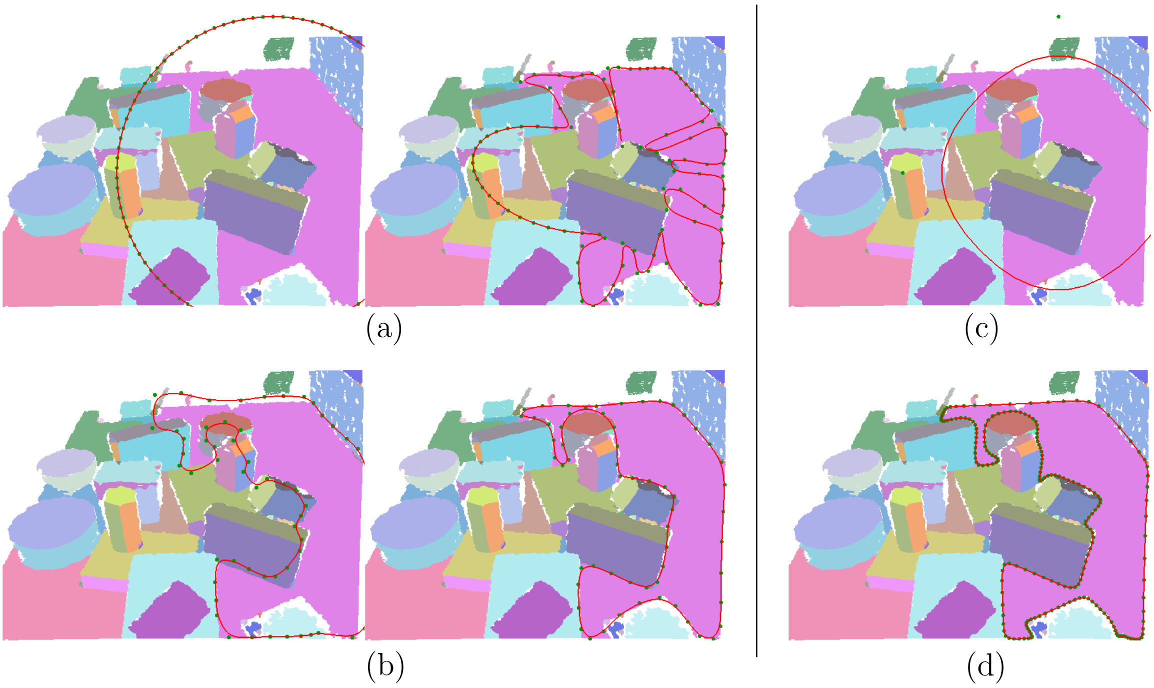


Figure 2.22: Comparison: (a) Initialization using the bounding circle. (b) PD, TD and SD disturbed by a single outlier in the middle of the dataset. (c) Our method applied to PD, TD and SD according to Equation (2.13) after convergence.

points. Note that the number of control points can sometimes be unnecessarily high as we are inserting knots at each iteration. Alternatively we could insert knots only when the curve converged with its current number of control points, but this would slower the overall convergence of the algorithm.

Curve fitting for reconstruction: Similar to [28] we have applied our approach to the task of reconstruction. As shown in Figure 2.24 a B-spline surface is fitted to the 3D point cloud. After that our method is used to determine the boundary, which allows for trimming the surface. Both, curve fitting and trimming is done in the parametric domain of the surface. Note that it is not necessary to compute the boundary points, since the AD de-weights points inside the curve, which means that our approach is also appropriate for dense point clouds. However, fitting a curve to a dense point cloud is much slower due to the high number of data points.

Robustness: We are using the same set of parameters for a certain type of point data (e.g. the Berkeley dataset) and do not need to adjust them for each image. Figure 2.25 shows fitting results with changing data characteristics, namely the point density and noise. Our method fails if the accuracy for EAKI ε_a is lower than the distance between some adjacent points, which follows from the fact that fitting a concave boundary is an ill-posed problem. In other words EAKI and CF treat the gaps between points as concavity and try to fill them. However, changing the parameter ε_a changes this behavior and leads to a satisfying solution (Figure 2.26).


(a) (c)

(b) (d)

Figure 2.23: Comparison: (a) *Squared Distance* (SD) with initialization using the bounding circle and a sufficient number of control points. (b) SD with manual initialization. Note that the accuracy at corners is quite low, due to a too small number of control points. Only if the shape is roughly estimated with a sufficient number of control points SD produces similar results as our approach. (c) Our method with initialization using the bounding circle. (d) Our method after convergence. Note how at points of sharp turns additional control points are inserted by EAKI.

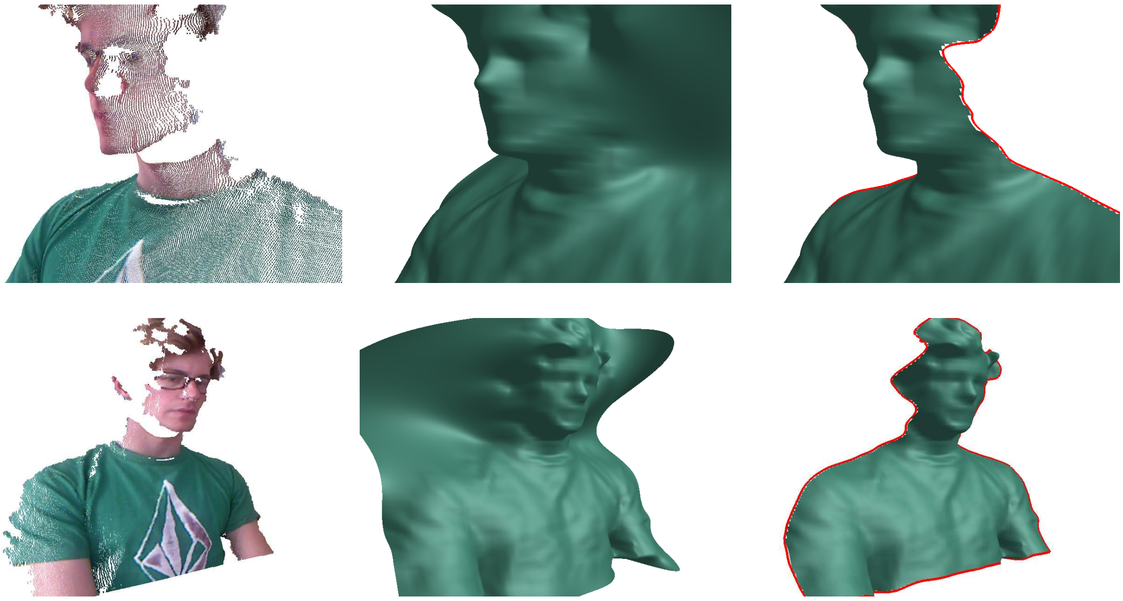


Figure 2.24: Reconstruction: The curve is fitted to a point cloud (left) in the parametric domain of a B-spline surface (middle) which allows for trimming it (right). ($\sigma = 5e^{-4}$, $\varepsilon_d = \varepsilon_a = 0.015$, $w_s = 0.5$, $w_c = 1.0$, $\gamma = -1.0$, 15 iterations).

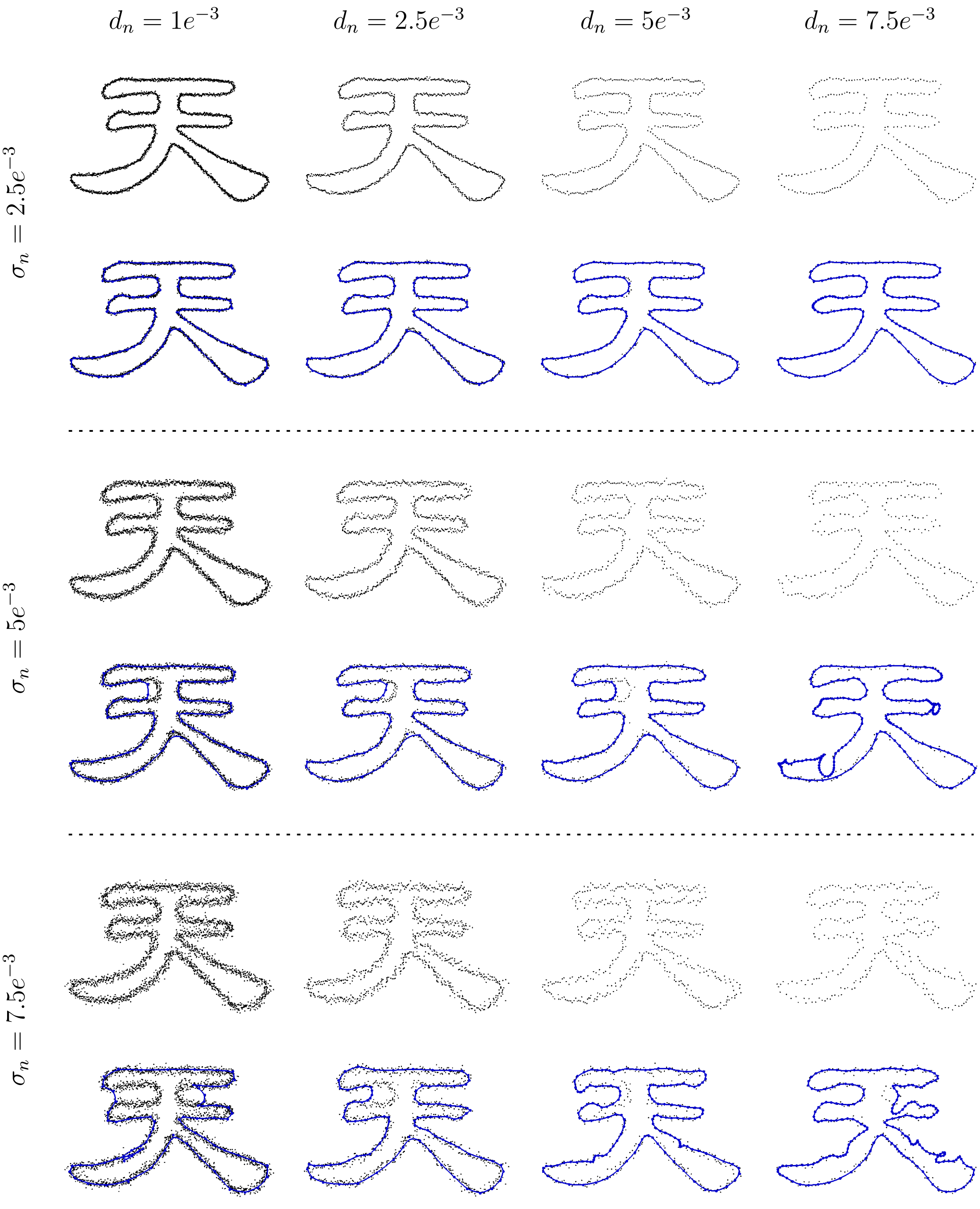


Figure 2.25: Robustness when fitting the Chinese character *tian*, without changing the parameters but the distribution of the data points. Point distance d_n increases from left to right, noise increases from top to bottom. The resulting curve is shown in blue after 40 iterations. ($\sigma = 0.0002$, $\varepsilon_d = 0.0$, $\varepsilon_a = 0.015$, $w_s = 0.5$, $w_c = 1.0$, $\gamma = -1.0$)

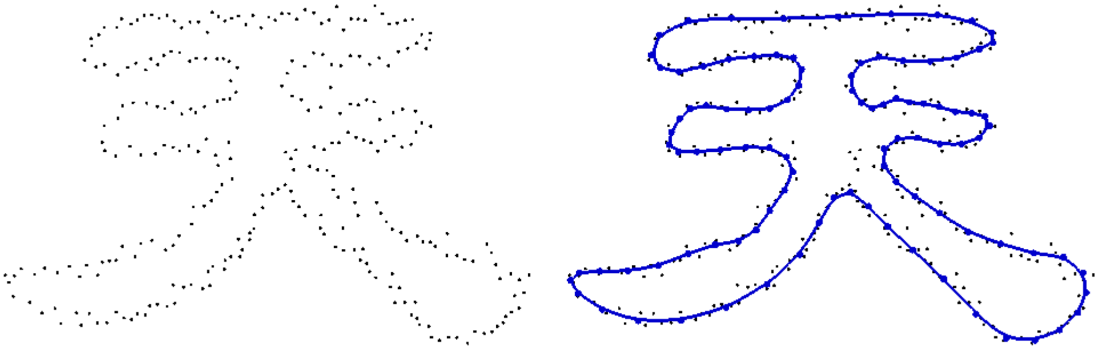


Figure 2.26: Fitting of the difficult case in the lower-right of Figure 2.25 ($\sigma_n = 7.5e^{-3}$, $d_n = 7.5e^{-3}$) with changed parameters ($\sigma = 0.0002$, $\varepsilon_d = 0.0$, $\varepsilon_a = 0.017$, $w_s = 0.5$, $w_c = 1.0$, $\gamma = -1.0$, 40 iterations).

2.6.2 Surface segmentation

We evaluate our approach using the online available *Object Segmentation Database* (OSD) [83], consisting of 110 table top scenes with various kinds of objects and with different complexities of scenes. The dataset provides RGB-D data with hand-annotated ground truth for all objects. Pre-segmentation, model parametrization and model refinement is evaluated with respect to over- and under-segmentation of surface patches on objects. Since the OSD provides ground truth for segmented objects rather than continuous regions, segments from our approach are assigned to an object when more than half of it overlaps. N_{true} and N_{false} are the number of correct and incorrect segmented points and N_{all} the number of all points on an object. Over-segmentation F_{os} and under-segmentation F_{us} is then defined as:

$$F_{os} = 1 - \frac{N_{true}}{N_{all}} \tag{2.23}$$

$$F_{us} = \frac{N_{false}}{N_{all}} \tag{2.24}$$

For a sequence of images with multiple objects in a scene, N_{true}, N_{false} and N_{all} are summed up over all frames. Figure 2.27 shows an object and its segmentation as rectangular region, where the different areas indicate the measures for over- and under-segmentation.

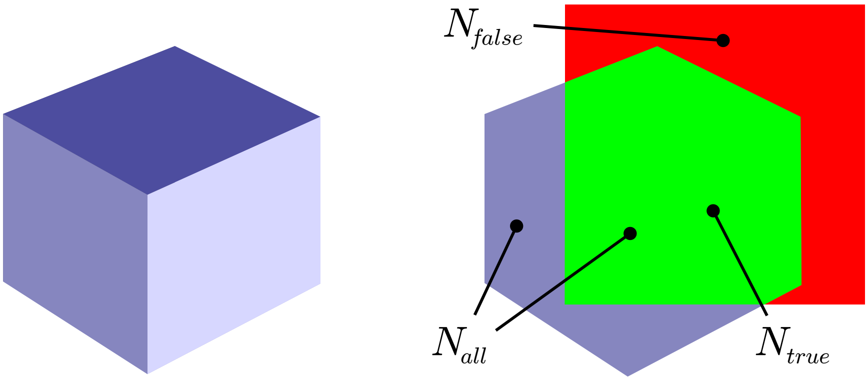


Figure 2.27: Correct (N_{true}) and incorrect (N_{false}) segmentation regions with respect to the ground truth (N_{all}). These values are used for defining over- and under-segmentation.

Pre-segmentation

Figure 2.28 shows two examples of pre-segmentation with two sets of parameters. We chose the parameters such that the image is slightly over-segmented, since wrongly split surfaces are merged during model selection. On the other hand, under-segmented regions including two different surface patches can not be split up any more. For a more detailed evaluation of pre-segmentation we want to point the interested reader to the work of [73].

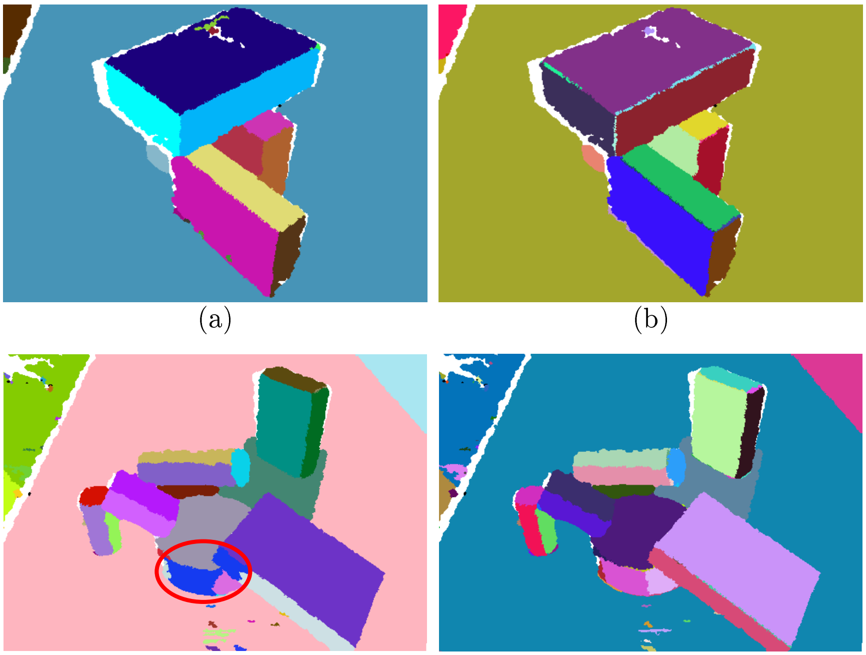

(a) (b) (c) (d)

Figure 2.28: Pre-segmentation of table top scene from OSD. While for image (a) the parameters are optimal, (b) shows additional patches on the edges of the box on the top. On the other hand, the same set of parameters causes under-segmentation on image (c) (blue patch) while (d) shows better pre-segmentation.

Model selection

During the model selection procedure, described in Section 2.3.2, neighbouring planar surface patches are getting merged. Reduction of patches through merging is highly dependent on the used images, because it usually happens on curved surfaces that planar pre-segmented patches get merged to a bigger B-spline surface. However, typically neighbouring patches of the same object get merged, if no discontinuities are in between.

Model refinement

Model refinement, described in Section 2.2.3, improves the boundary of segmented patches using colour information. Further depth data not available is filled by mapping the point from image space to a 3D surface. This leads to the following improvement with respect to segmentation accuracy: The value for over-segmentation decreases about 36% (from 3.92% to 2.50%) and under-segmentation about 30% (from 1.71% to 1.17%).

Figure 2.29 shows the sequential steps of our segmentation framework. RGB-D images are pre-segmented to planar patches, which are then merged with respect to curved surfaces during model selection. The refinement step fills missing data and adjusts depth and colour edges.

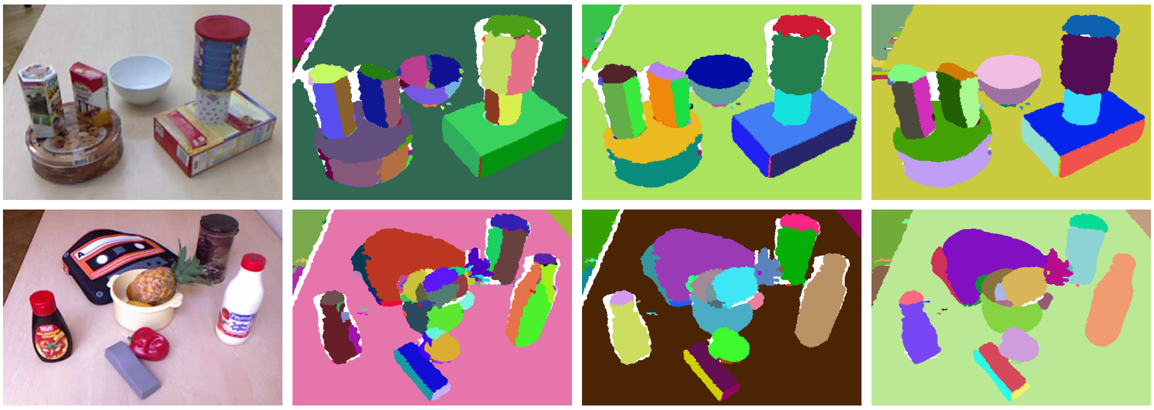


Figure 2.29: From left to right: RGB-D image from the Microsoft Kinect, pre-segmentation, model selection and model refinement.

2.6.3 Object segmentation

The object segmentation was again evaluated using the OSD consisting of 110 RGB-D images differentiated into six subsets with different complexity:

- Boxes
- Stacked Boxes
- Occluded Objects
- Cylindrical Objects
- Mixed Objects
- Complex Scene

45 images were used for learning the SVM for grouping the parametric surfaces (Section 2.4.1). Figure 2.30 shows the different kinds of scenes and the results achieved with our object segmentation algorithm.

Table 2.2: Results of object segmentation. SVM accuracy, over- and under-segmentation.

Set	SVM	F_{os}	F_{us}
Boxes	98.19%	0.2%	17.2%
Stacked Boxes	98.99%	0.0%	28.2%
Occluded Objects	99.23%	0.0%	0.2%
Cylindrical Objects	96.77%	2.6%	3.5%
Mixed Objects	94.97%	1.3%	39.2%
Complex Scene	98.97%	5.4%	145.5%
MEAN	98.41%	2.7%	69.5%

The evaluation results from the six different trainings sets are shown in Tab. 2.2. The first column presents the accuracy of the SVM predictions for relations between neighbouring surface patches, followed by the over- and under-segmentation. Note that even occluded objects are successfully segmented by using two SVMs, where one is specialized to resolve such difficult situations. More details on the two stage SVM algorithm are available in the work of [83].

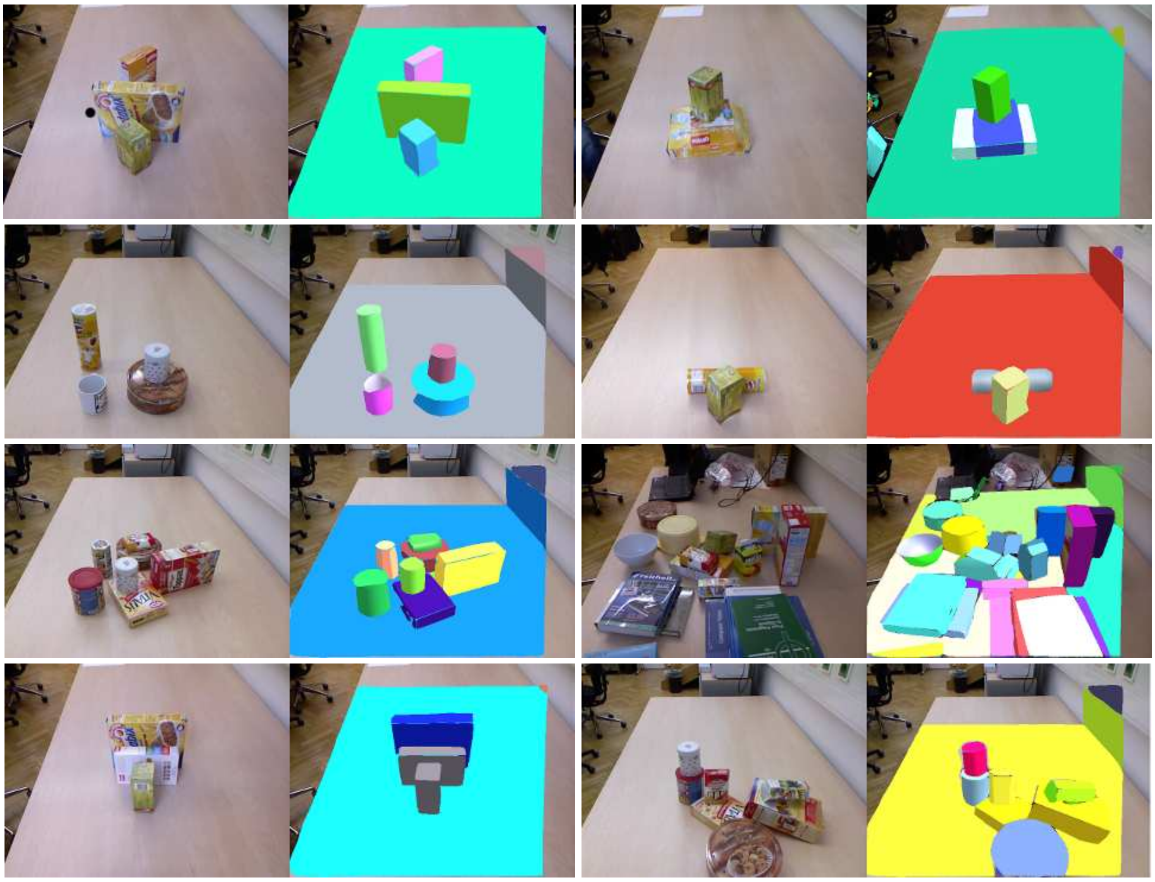


Figure 2.30: Selected examples of segmented objects with the proposed approach (objects randomly coloured). The top six examples showing results from each dataset, the bottom row is showing under-segmentation caused by wrong predictions.

2.7 Discussion

In this chapter we have introduced several methods with the goal of object segmentation and reconstruction. We do not claim to solve these problems in a general way, but instead tested methods and models that allow to operate in a cognitive robotic environment characterized as outlined in Chapter 1. The developed algorithms contribute to various fields of research.

As no usable curve fitting algorithm for scattered non-dense point-clouds exists we present the method of Section 2.2 to the community. Unfortunately this method is not completely generic and six parameters need to be tuned to the sensor and scene used. Considering that finding the non-convex hull of an unorganised point-cloud is an ill-posed problem, we only consider the estimation of these parameters as future work and not their removal.

Object segmentation on a range image is not a new problem. However, due to the recently available cheap RGB-D sensors this field of research received a lot of attention in the past years. Many algorithms have been published to robustly segment and reconstruct objects in real-time. Our approach puts the focus on a meaningful representation of these objects, which is why we choose to model them using parametric geometric primitives such as B-splines. This description gives additional information for object categorisation, grasping and reverse engineering.

The methods described in this chapter takes advantage of the tracking algorithm and the TSD introduced in the following chapter. Segmented objects of interest are selected and visually tracked. TSD is used to identify good views from which new shape information is learned. Therefore new surface segments are attached to the object or, when it overlaps with an existing surface, merged accordingly. This consecutively improves and extends the shape model. The drawback of our reconstruction approach is that it neither produces watertight models nor does it model local details due to the limited geometric primitives we are using for model selection. The possibility for finding the intersections and seams of the object model is given especially when taking colour information into account. Also there exist generalisations of B-spline surfaces which would allow for local refinement by knot insertion (e.g. T-spline surfaces) in contrast to global refinement as in our approach. However, since the statement of this thesis in this context is to benefit from parametric models and to improve them in the cognitive sense we want to postpone this problem to future work.

At this point the framework presented in this thesis so far is already an excellent example of knowledge fusion from different cues, tightly welding the components together. In this chapter the shape learning scheme relies on visual tracking to fuse different views, whereas the tracking algorithm itself relies on the shape model learned so far. Starting from scratch a single view is sufficient to start tracking which in return allows for shape modelling. The appearance model is built up and refined in a parallel manner with hardly any prior knowledge or constraints. As we will see in the subsequent chapters this characteristic applies in the very same manner. This is what we found to be the core principle for cognitive robotics depicted in Figure 1.5.

Chapter 3

Colour based object tracking

Once a partial model of the object appearance, namely its shape and colour, has been learned, we can visually track it during pose changes. Subsequently we identify new good views and, as shown in the previous chapter, add the new information to obtain a more and more complete appearance model.

For learning about physical behaviour of objects as in Chapter 4 and in more detail in [47, 71, 48, 25] the robot has to observe its motion (Figure 3.1, middle). Again accuracy of tracking and also detecting if the object is tracked correctly are important to decide whether a certain trajectory should be taken into account for learning. For a robot operating in a complex unpredictable environment, the challenge is to develop a tracking method that is robust to different lighting conditions, partial occlusion, and motion blur.

Today this is achieved best by model-based tracking of objects and numerous solutions using different feature types, models and mathematical frameworks have been developed, where today's computational power allows for several real-time solutions. However, practical application of these methods is often limited for various reasons. For example, some methods report good results, without giving actual numbers on accuracy [9, 68, 42, 66]. Others are capable of handling partial occlusion or changing lighting conditions [66, 95, 70, 94] but can not differentiate between deteriorating tracking conditions and lost tracks. Some methods are restricted in their degrees of freedom, e.g. 140 degrees of rotation as in [70], require off-line learning [95] or are limited to either textured [89, 78] or low- textured objects [97]. Also recovery from lost tracks is rarely handled with a few exceptions [78, 89], which are tracking-by-detection approaches.

Another requirement in robotics is computational efficiency to react to observed situations in time. Consider again the grasping scenario, where we want to use visual servoing to adapt the grasping movement on-line. Hence, we require real-time performance, i.e. processing time within the frame rate of a typical camera (25-50 Hz). For now we are using RGB data only, since at the time of development no range sensors of sufficient resolution achieving such a frame rate were commonly available.

To meet all these requirements we propose to tackle the core problem of de-

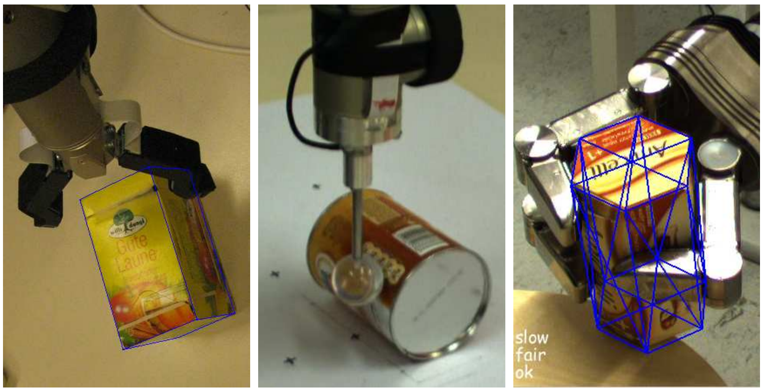


Figure 3.1: Tracking for robotic applications. Left: grasping; middle: learning about object motion; right: grasp stability.

tecting tracking failure and take advantage of this supervisory knowledge to achieve automatic object tracking using texture mapping, pose recovery and online learning. Hence, the approach is based on the following methods:

- *Tracking-state-detection (TSD):* To know whether we are tracking correctly, whether the object is occluded or whether we lost track we employ our novel TSD method. The knowledge of the tracking state, including speed and confidence of tracking, allows for triggering online learning or pose recovery.
- *Texture mapping:* We take advantage of texture, if available, to boost robustness of tracking, especially in cluttered scenes.
- *Pose recovery:* To initialise tracking and recover lost tracks we use distinctive features placed on the surface of the object model.
- *Online learning:* We learn these feature points and surface texture of the object automatically while tracking.
- *Model completeness:* A probabilistic formulation allows to reason if sufficient information of the object has been gathered.

Additionally to tracking itself we learn colour information and map it on the surface of the shape model we obtained so far. Since this mapping is a trivial task we do not spend a section on colour learning on its own, but rather focus on how to identify good views and complete the model.

Overview: Section 3.1 gives an overview of related work on visual tracking algorithms. In Section 3.2 we formulate tracking as particle filtering using a modified version of the Bootstrap filter [34] and show how to draw observations by projecting the model into image space. Section 3.3 introduces TSD which allows to reason about the current tracking quality, convergence and whether tracking has lost the object or is occluded. We show how surface texture and SIFTs of a tracked object can be learned online and how they are used for re-detection. In Section 3.4 we evaluate our approach with respect to the requirements established above. Figure 3.2 gives a high-level overview of our tracking system.

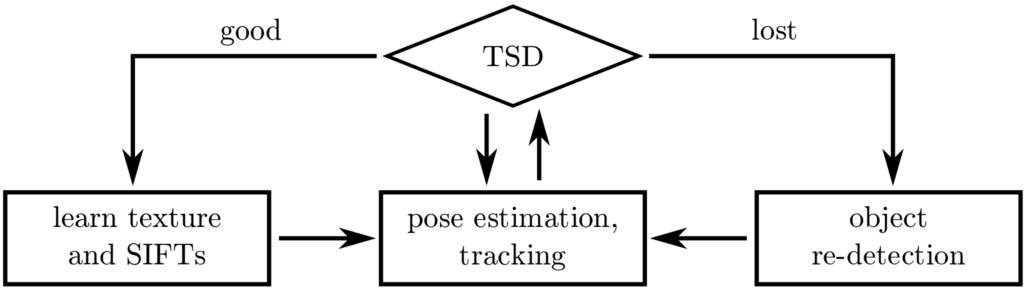


Figure 3.2: Tracking an object with the tracking-state-detection (TSD). Good views are used for learning texture and SIFTs. When tracking fails and the object is visually lost, pose recovery is triggered to re-detect it.

3.1 Related work

Tracking the pose of an object by analysing a stream of TV images in real-time goes back to the early eighties [77, 26]. One of the first successful approaches of tracking objects based on edges was the RAPiD system [36]. It used points on model edges and searched for corresponding image edges the edge gradient. Subsequent approaches aimed at improving robustness in tough real-world scenarios [22, 9, 68, 95, 43]. Approaches based on globally matching model primitives with primitives extracted from the camera image [60, 32, 44, 49, 86] have been used for applications such as robot and car tracking, but were later replaced by improved versions of the RAPiD type.

[66] also use edges and textures for tracking. Their approach extracts point features from surface texture and use them, together with edges, to calculate the object pose. This turns out to be very fast as well as robust against occlusion. Our approach not only uses patches but the whole texture, which usually lets the pose converge very quickly to the accurate pose. Since the algorithm runs on the GPU, it is as fast as the method in [66]. The work presented in [97] uses edge features to track but does not take into account texture information. This makes it less robust against occlusion. Since the search area in that approach is very small, it is also less robust against fast movement and gets caught in local minima.

More recent approaches aim to solve most of the problems of tracking, such as [94] where the authors are matching the camera image with pre-trained keyframes and then minimizing the squared distance of feature points taking into account neighbouring frames. The approach described in [70] uses a modified version of the Active Appearance Model which allows for partial and self occlusion of the objects and for high accuracy and precision. In [17] the authors minimize the optical flow resulting from the projection of a textured model and the camera image. To compensate for shadows and changing lighting they apply an illumination normalization technique.

In [50] the authors introduce real-time tracking to robotic manipulation. They are using the method proposed in [64], where they project the CAD model into image space, and try to minimize a cost functional for the distance to image edges found along the gradients of the edges of the model. The work presented in [29] describes an approach for real-time visual servoing using a binocular camera setup to estimate the pose by triangulating a set of feature points. Similar to our approach, [89] takes advantage of robust Monte Carlo particle filtering to determine the pose of the camera with respect to SIFT features, which are localized in 3D using epipolar geometry.

Missing in all methods is to detect when tracking fails rather than reporting tracking trapped in a local optimum. The proposed TSD tries to solve this and we develop the approach to make it work automatically.

3.2 Pose estimation

The full 6 DOF pose of the object is identified by using colour and edge information from shape and texture. We project a model, typically consisting of triangles or quads with attached texture, into image space and compare it against the camera image. The pose is estimated using a modified version of the Sequential Importance Resampling (SIR) particle filter [19]. Image processing methods such as Gaussian smoothing and edge extraction as well as pixel-wise comparison of the projected model is accelerated using a typical graphics processing unit (GPU).

3.2.1 Transformations on the Euclidean group

Visual observation of the trajectory of the object is the problem of finding the transformations $\mathbf{T}_t$ given a sequence of images I_t, sampled over the time. Since we constrain the tracking approach to rigid objects, the trajectory can be described as transformations on the Euclidean group $SE(3)$. These are represented as

$$\mathbf{T}(\mathbf{x}) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.1}$$

where $\mathbf{R}(\boldsymbol{\theta})$ is a rotation matrix and $\mathbf{t} = [x, y, z]^T$ a translation respectively. Rotations are realised using unit quaternions $\mathfrak{q}$ with $||\mathfrak{q}|| = 1$, which constrains $\mathbf{R}$ to be an element of the $SO(3)$. They provide a simple way to represent uniform axis-angle rotations and avoid the gimbal lock which occurs when trying to model rotations by Euler angles. Quaternions are extensions to the complex numbers,

$$\mathfrak{q} := r + \theta_x \mathfrak{i} + \theta_y \mathfrak{j} + \theta_z \mathfrak{k} \tag{3.2}$$

conveniently written as

$$\mathfrak{q} := r + \boldsymbol{\theta} \tag{3.3}$$

$\mathfrak{i}$, $\mathfrak{j}$, $\mathfrak{k}$ are imaginary units satisfying $\mathfrak{i}^2 = \mathfrak{j}^2 = \mathfrak{k}^2 = \mathfrak{i}\mathfrak{j}\mathfrak{k} = -1$. A rotation by α radians about the axis $\mathbf{u}$ is defined as quaternion by

$$\mathfrak{q} := \cos(\alpha/2) + \mathbf{u}\sin(\alpha/2)$$

Let $\mathbf{a}$ be an ordinary vector in $\mathbb{R}^3$ represented as quaternion with its real value $r = 0$, then a rotation of this vector is simply the quaternion product.

$$\tilde{\mathbf{a}} = \mathfrak{q}\mathbf{a}\mathfrak{q}^{-1} \tag{3.4}$$

Since for unit quaternions one of the values of the quaternions is always given by the other three. Together with the translation $\mathbf{t}$ this results in a state vector $\mathbf{x} = [x, y, z, \theta_x, \theta_y, \theta_z]^T$ of 6 DOF.

3.2.2 Monte Carlo particle filtering (MCPF)

A particle filter, such as the SIR (Sequential Importance Resampling) or the Bootstrap filter, explained in [20] and more detailed in [19], estimates the current state $\mathbf{x}_t$ based on the previous state $\mathbf{x}_{t-1}$ and the current observation $\mathbf{y}_t$.

$$\begin{aligned} \mathbf{x}_t &= f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) + \mathcal{N}_{t-1} \\ \mathbf{y}_t &= g(\mathbf{x}_t) + h(\mathbf{x}_t) \end{aligned} \tag{3.5}$$

For now we assume a static motion model, without taking into account external forces $\mathbf{u}_{t-1}$, yielding $f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{x}_{t-1}$. The observation $g(\mathbf{x}_t)$ is based on the image gradients, whereas $h(\mathbf{x}_t)$ is based on the hue component of the HSV (hue, saturation, value) colour space. Figure 3.3 and Algorithm 2 show the behaviour of a Monte Carlo particle filter which sequentially resamples and replaces the particles depending on their weights.

In the initial phase (1a) the particle distribution is initialised using a pose $\mathbf{x}_0$ given by user input or by a feature based object detection system as described in Section 3.3.3. The particles are sampled from the normal distribution $\mathcal{N}(\mathbf{x}_0, \boldsymbol{\sigma}_b)$. The confidence value c_0 is set to 1. According to Equation (3.6) this leads to an initial variance of $\boldsymbol{\sigma}_0 = 0$ and therefore to no movement at all in step (2a). Note, that during tracking (i.e. $t \geq 1$) the confidence value c_t is typically below 1.

Given the observations $\{\mathbf{y}_t | t \in \mathbb{N} \cup \{0\}\}$, our aim is to estimate the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_t)$. $\mathbf{y}_t$ corresponds to the current image given by a camera sensor. In step (2b) for all poses $\mathbf{x}_t^i$ the importance weights are evaluated, approximating the probability distribution of observations $p(\mathbf{y}_t|\mathbf{x}_t)$. The posterior distribution is given by the *Bayes' theorem*.

The key idea of the MCPF lies in the approximation of $p(\mathbf{x}_t|\mathbf{y}_t)$ with a discrete distribution $P_N(\mathbf{x}_t|\mathbf{y}_t)$. Particles with low weights are eliminated, whereas the ones with high weights are multiplied. The final pose reported by the tracker is the weighted mean of the best $\overline{N} < N$ particles, $\overline{\mathbf{x}}_t$. This is the classical Bootstrap filter, introduced by [34], which is typically applied for visual tracking as it has several advantages. First, it is very easy to implement. Second, the algorithm can be efficiently executed in parallel which we exploit using the GPU. And third, it is to a large extent modular which allows to replace certain steps by more sophisticated methods as follows.

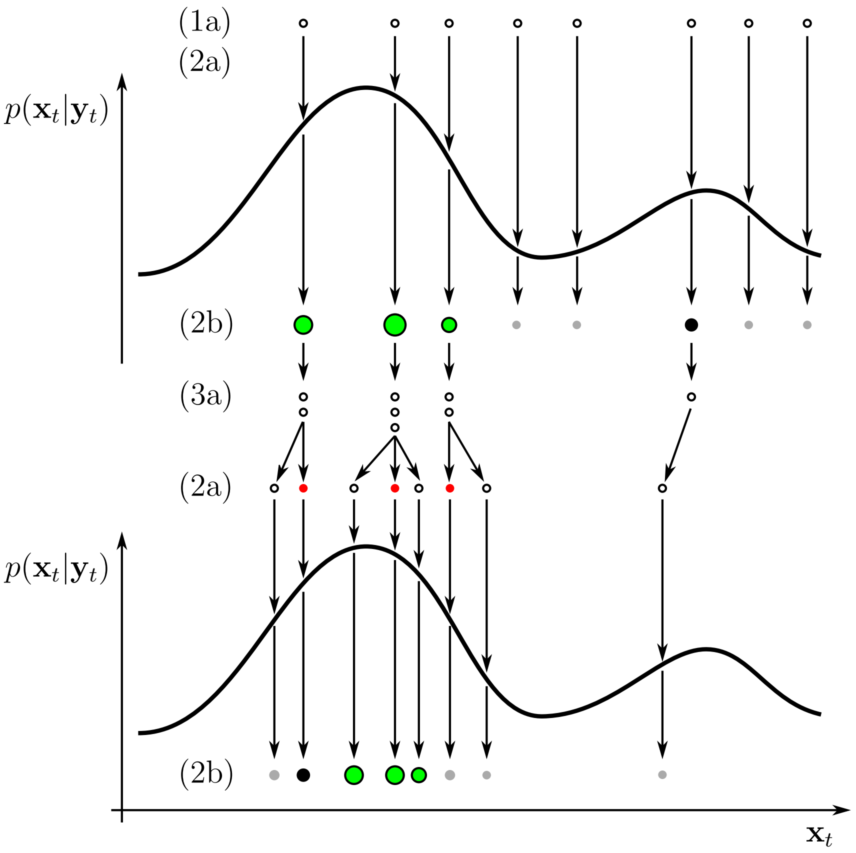

(1a)
(2a)
$p(\mathbf{x}_t|\mathbf{y}_t)$
(2b)
(3a)
(2a)
$p(\mathbf{x}_t|\mathbf{y}_t)$
(2b)
$\mathbf{x}_t$

Figure 3.3: (1a) The MCPF starts with a uniformly weighted distribution of particles. (2b) The weight for each particle is evaluated, which results in an approximated distribution. (3a) According to the importance weights the selection step assigns weak particles (grey) to the fittest (green). (2a) Then all particles are moved using Gaussian noise except one, whose pose is fixed (red). Afterwards the weights are evaluated again closing the loop. (The labels correspond to Algorithm 2.)

Confidence dependent noise: In the sampling step (2a) of Algorithm 2, we adjust the amount of the system noise $\mathcal{N}$ according to the confidence of the previous tracking step c_{t-1}. This means that as the confidence of the particles increases, their degree of distribution decreases, leading to faster convergence and less jitter. Given the requirements for tracking accuracy and speed for a typical table top scenario we chose a basic standard deviation $\boldsymbol{\sigma}_b = [\sigma_x, \sigma_y, \sigma_z, \sigma_{\theta_x}, \sigma_{\theta_y}, \sigma_{\theta_z}]^T$ with $\sigma_{x,y,z} = 0.03$ m for the translational and $\sigma_\theta = 0.5$ rad for the rotational degrees of freedom.

Fixed particle poses: Since we want to perform in real-time we use a limited number of particles which causes jitter of the final pose $\overline{\mathbf{x}}_t$. At the same time σ_t is never 0 ($c_t < 1$). This means that the best $\overline{N}$ particles will disperse around the true pose. With a sufficiently large number of particles, this would not be a problem, but due to our small number of particles it results in visible jitter. Now we could increase the number of particles sacrificing real-time performance or use the following

Algorithm 2 Bootstrap filter, modified with respect to importance sampling.

1. Initialisation
    (a) For $i = 1, \ldots, N$, sample $\mathbf{x}_0^i \sim p(\mathbf{x}_0) \sim \mathcal{N}(\mathbf{x}_0, \boldsymbol{\sigma}_b)$ and set $c_0 = 1$, $t = 1$.
2. Importance sampling
    (a) For $i = 1, \ldots, N$, sample $\tilde{\mathbf{x}}_t^i \sim p(\mathbf{x}_t | \mathbf{x}_{t-1}^i, c_{t-1})$ with

    $$p(\mathbf{x}_t^i | \mathbf{x}_{t-1}^i, c_{t-1}) \sim \mathcal{N}(\mathbf{x}_{t-1}^i, \boldsymbol{\sigma}_{t-1}^2), \qquad \boldsymbol{\sigma}_{t-1} = (1 - c_{t-1}) \boldsymbol{\sigma}_b \tag{3.6}$$

    (b) For $i = 1, \ldots, N$ of $\tilde{\mathbf{x}}_t^i$, evaluate the confidences and the overall confidence using Equation (3.13). Normalize the confidence values for the importance weights

    $$w_t^i \sim p(\mathbf{y}_t | \tilde{\mathbf{x}}_t^i), \qquad w_t^i = \frac{c_t^i}{\sum_{i=0}^{N} c_t^i} \tag{3.7}$$
3. Selection step
    (a) Resample with replacement N particles $\mathbf{x}_t^i$ from the discrete distribution

    $$P_N(\mathbf{x}_t | \mathbf{y}_t) = \sum_{i=1}^{N} w_t^i \delta(\tilde{\mathbf{x}}_t^i) \tag{3.8}$$

    (b) Set $t = t + 1$ and go to step 2

---

heuristic. The idea is to keep the pose of the best particles fixed instead of sampling from $\mathcal{N}$. In detail, for each set of particles, with the same posterior $\tilde{\mathbf{x}}_t^i$, one is chosen where no noise is applied. Obviously this only makes sense if there are more than one particles in the set. The red particles in Figure 3.3 indicate the set where the pose is fixed which we denote by $\mathbf{X}_t^f$. This ensures convergence, efficiently reduces jitter and increases robustness of tracking as shown in Figure 3.4.

Iterative particle filtering: As proposed in previous works [75] and [84], iterative particle filtering increases responsiveness to rapid pose changes. Therefore steps 2 and 3 of Algorithm 2 are performed several times on the same image. This means that the poses of the particles are iteratively shifted to the peak of the distribution. In contrast to pure one-time re-weighting of the existing particles this leads to a better approximation of the distribution $p(\mathbf{x}_t | \mathbf{y}_t)$ per image. Figure 3.5 shows the improvement over conventional particle filtering when using 8 iterations with 100 particles each vs. 1 iteration with 800 particles. The iterative version follows the object motion

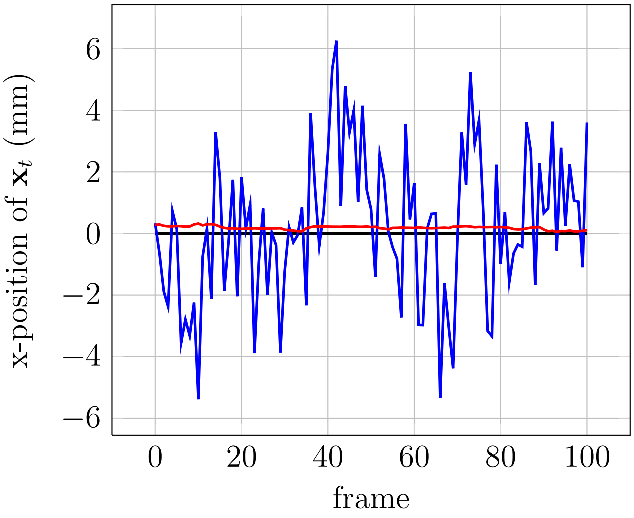


Figure 3.4: Visible jitter of the pose $\overline{\mathbf{x}}_t$ (blue) and improvement when fixing the pose of the best particles (red).

much faster.

3.2.3 Image processing and confidence evaluation

At time-step t for each particle i, we project the model of the object into the image space using the transformation $\mathbf{T}^i$. For simplicity we skip t in the mathematical formulations since the following equations are computed in the same time-step. The geometry of the model is defined by vertices and faces. The texture, i.e. colour of the model is aligned to the faces by employing UV mapping, a standard technique of computer graphics. In image space we compute the colour gradients of the model $\mathbf{g}_M^i$ and of the image captured by the camera $\mathbf{g}_I^i$, where $\mathbf{g} \in \mathbb{R}^2$. For each point (u, v) on the model M in image space we can compute the difference between both of the gradients at that position, by superimposing the projected model over the image. The match m_g^i of a particle is defined as the sum of the differences of the gradients, and s_g^i is a normalising constant given by the sum over all model gradients.

$$\begin{aligned} m_g^i &= \textstyle\sum_{(u,v)\in M} \left| \mathbf{g}_M^i(u,v) - \mathbf{g}_I^i(u,v) \right| \\ s_g^i &= \textstyle\sum_{(u,v)\in M} \left| \mathbf{g}_M^i(u,v) \right| \end{aligned} \tag{3.9}$$

Additionally to the difference of gradients the colour defined in HSV (Hue, Saturation, Value) space is used for matching. Analogous to Equation (3.9) the match for colour m_h^i and its normalising constant s_h^i are defined as

$$\begin{aligned} m_h^i &= \textstyle\sum_{(u,v)\in M} \left| h_M^i(u,v) - h_I^i(u,v) \right| \\ s_h^i &= \textstyle\sum_{(u,v)\in M} \left| h_M^i(u,v) \right| \end{aligned} \tag{3.10}$$

To achieve invariance with respect to brightness the hue values are used for matching the projected model h_M^i and the image h_I^i. The advantage of using colour based

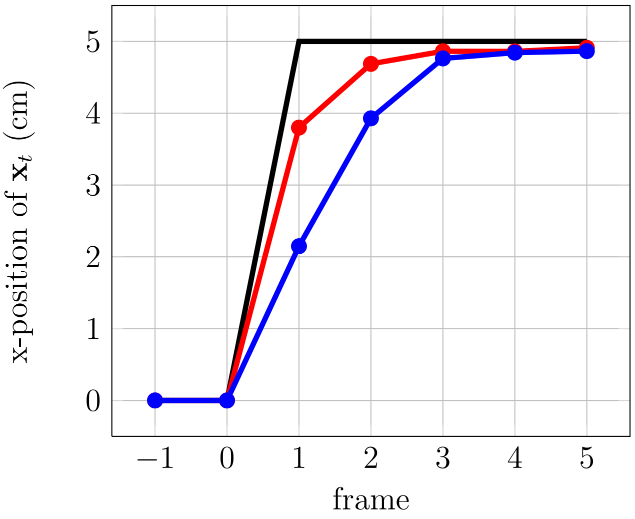


Figure 3.5: Step response showing the faster convergence of iterative- (red, 8x100) against conventional particle filtering (blue, 1x800). Both are using the same number of particles.

tracking is the increase of robustness against edge based clutter. Of course it is less robust against changing lighting but the combination of both kinds of cues can significantly improve the overall performance. The confidence of a particle $\mathbf{x}^i$ for matching gradients c_g^i and colour c_h^i are defined as

$$\begin{aligned} c_g^i &= \tfrac{1}{2}\left(\frac{m_g^i}{s_g^i} + \frac{m_g^i}{\frac{1}{N}\sum_{j=1}^{N} s_g^j}\right) \\ c_h^i &= \tfrac{1}{2}\left(\frac{m_h^i}{s_h^i} + \frac{m_h^i}{\frac{1}{N}\sum_{j=1}^{N} s_h^j}\right) \end{aligned} \tag{3.11}$$

where the first term is the match normalised with respect to s_i. The second term is normalised with respect to the mean over all particles, de-weighting particles with a low number of pixels. This prevents the system from getting stuck in poses with a small number of pixels. The combined confidence of a particle is the product of the gradient- and colour confidence.

$$c^i = c_g^i c_h^i \tag{3.12}$$

The overall confidence of the current observation t is defined by the arithmetic mean of the confidences of all particles i in the distribution.

$$c_t = \frac{1}{N}\sum_{i=1}^{N} c^i \tag{3.13}$$

Figure 3.6 shows our implementation of the tracking system. The pose is refined using iterative particle filtering until new data arrives from the image capturing pipeline. If this happens, the image edges are updated. Otherwise the model is transformed according to the particles at frame $t-1$. The edges of the model are extracted and matched with the current image edges. Subsequently the weights are updated and the particles are re-sampled with replacement.

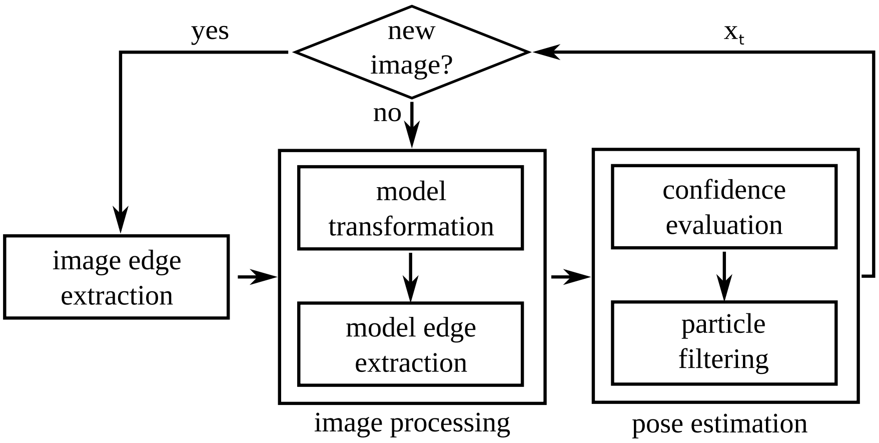


Figure 3.6: Tracking by iterative particle filtering.

3.3 Learning

Starting from a purely geometric representation of the object to track, robustness is improved by adding colour texture and feature based information. Considering a cognitive robotic environment, with as little user-input as possible, the key to automatically update the object representation is to detect the current state of tracking. This allows to identify good views for updating and improving the model representation. Furthermore a quantitative measure of completeness of the model is necessary to determine views that have not been learned so far or where enough information is already available.

3.3.1 Tracking-state-detection (TSD)

As outlined above observing the current state of the tracker is important for assessing the validity of the output as well as allowing to trigger recovery from lost tracks. TSD is a mechanism that indicates convergence, quality and overall state. It requires to reliably detect,

- if the object is moving,
- if the algorithm converged,
- if a good view for learning features is achieved,
- if the object is occluded,
- and if the algorithm is actually tracking the correct object or got caught in a local maximum.

Furthermore TSD not only allows for learning about the object, it is also beneficial for tasks like pose recovery, robotic manipulation, visual servoing, learning physical behaviour from visual observations and so forth.

Convergence rate: The convergence rate is important to determine if the object is moving or still. This measure must be independent from the quality of the current observation which might be influenced by occlusion or modelling errors, which means that just looking at the confidence value c_t is not enough. Observing the speed of the trajectory is not satisfying for three reasons. First, the first derivative of the position amplifies noise. Second, it depends on the scale of the object and the point of view. And third, the elements of the speed vector derived from the position vector $\mathbf{x}_t$ are not of the same scale. Instead the fixed particles described in Section 3.2.2 are analysed. In more detail, the intersection and union of the set of fixed particles $\mathbf{X}^f$ at frame t and $t-1$ is computed.

$$\begin{aligned} \hat{\mathbf{X}}^f &= \mathbf{X}^f_t \cap \mathbf{X}^f_{t-1} \\ \check{\mathbf{X}}^f &= \mathbf{X}^f_t \cup \mathbf{X}^f_{t-1} \end{aligned} \tag{3.14}$$

The intersection represents the particles that did not move from one frame to the other. Then the mean of the weights of the particles in $\hat{\mathbf{X}}^f$ normalized with respect to the weights of the particles in $\check{\mathbf{X}}^f$ is an indicator of convergence.

$$\begin{aligned} v = &\ \textstyle\sum_{i=1}^{\hat{N}} \frac{w^i}{\sum_{j=1}^{\check{N}} w^j} \\ \text{with} &\ w^i(\mathbf{x}^i|\mathbf{x}^i \in \hat{\mathbf{X}}^f) \\ \text{and} &\ w^j(\mathbf{x}^j|\mathbf{x}^j \in \check{\mathbf{X}}^f) \end{aligned} \tag{3.15}$$

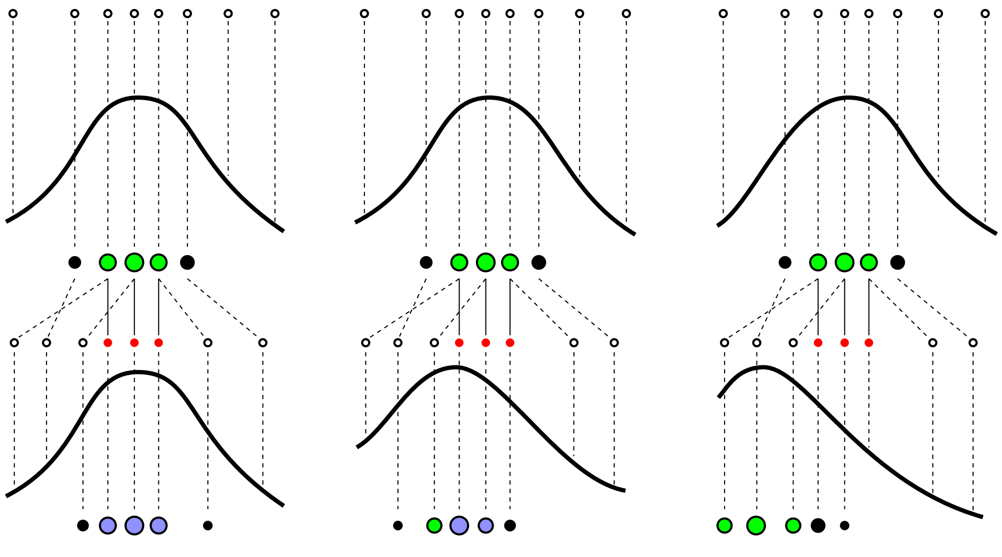
Figure 3.7 compares convergence in the case of no (static), slow- and fast movement of the underlying distribution.

Quality: To give a statement about the quality of the current pose we use the overall confidence c_t which corresponds to the match of a pose hypothesis to the image evidence. We classify this measure to obtain qualitative statements by applying thresholds to distinguish if tracking is *good*, *fair* or *bad* ($c_t > 0.5$, $0.5 \geq c_t \geq 0.3$ and $c_t < 0.3$ respectively).

Loss: Another task of TSD is to determine if the algorithm is tracking the object correctly or has been lost and got stuck in a wrong local maximum of the probability distribution. For Monte Carlo methods the effective particle size N_{eff} is introduced by [19]. Typically it is approximated by

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N}(w_t^i)^2} \tag{3.16}$$

and pose recovery is triggered when $\hat{N}_{eff}$ drops below the threshold $N_{thresh} = N/3$.



(a) static (b) slow (c) fast

Figure 3.7: Convergence rate for a static (left), slow- (middle) and fast (right) moving distribution. Green particles are the fittest. Red ones are within the set of fixed particles $\mathbf{X}^f$ according to the definition in Section 3.2.2. Blue ones are within the set of intersection $\hat{\mathbf{X}}^f$, from which the normalized mean weight is used for defining the convergence rate.

Occlusion: A little more tricky is to observe whether the object is occluded or not. Therefore a global histogram descriptor, taking into account edge- and colour information, is introduced. Similar to SIFT [61] gradients and hue values are sampled and accumulated into orientation histograms summarizing the contents over 5x5 partitions (Figure 3.8). This is done for the camera image and the projection of the model. Figure 3.9 shows the histograms of all the subregions and their intersection values respectively. This allows to determine how much of the object is occluded and which parts. Note that subregions which do not overlap the object sufficiently are not taken into account (e.g. top-left and lower-left subregion of Figure 3.9).

3.3.2 Texture mapping

Tracking is based on a CAD model which (initially) does not include surface texture. This is sufficient for non-textured objects, where all we can observe are edges resulting from occlusion and surface discontinuity. For textured objects, additional edges provided by the texture significantly improve robustness. The camera image provides the desired colour information of the object. The geometry of the object, i.e. the vertices are projected into image space to determine their alignment with respect to the texture. TSD is employed to select good views. Further only faces of the model are taken into account, which are approximately pointing in the opposite direction of the camera view vector (i.e. faces that are parallel to the image plane). For those faces the respective region of the camera image is cut out. The u, v-coordinates in

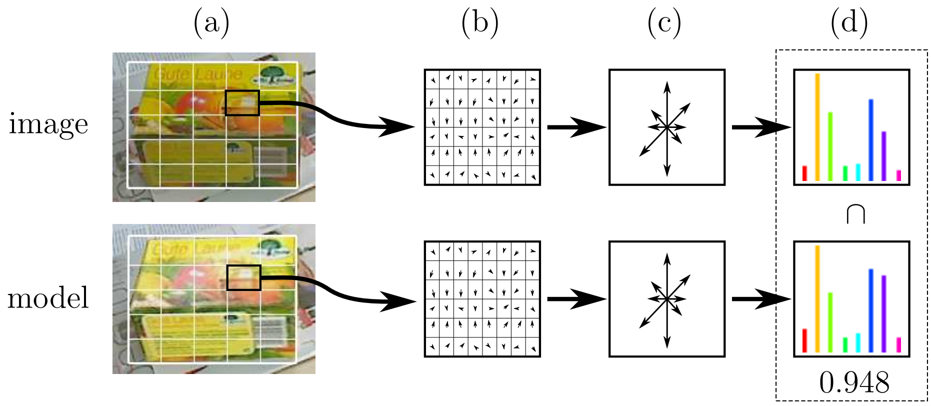


Figure 3.8: Histogram descriptor: The gradients and hue values of the subregions (a) are sampled (b) and accumulated into orientation histograms (c), both for the model and the image. The intersection of the histograms (d) represent the match of this specific subregion.

pixel space are calculated by projecting the vertices using transformation **T** provided by the tracker and the camera intrinsics.

3.3.3 SIFT mapping and object re-detection

While edges are well suited for fast tracking we use highly discriminating SIFT features for object detection (where again we use a GPU implementation [29]). Hence, we follow a standard approach similar to Gordon and Lowe [33] and Collet et al. [11] but our training phase differs in that we do not build a sparse 3D SIFT point model via bundle adjustment but use the 3D pose and object geometry already provided by the tracker. Therefore the view ray according to the u, v pixel coordinates of the SIFT points are calculated using the camera intrinsics. Then the view rays are intersected with the faces of the object model at the current pose $\mathbf{x}_t$ to get the 3D positions with respect to the object pose. SIFT features falling outside the object boundary are discarded.

To speed up object detection, SIFT features are represented using a codebook (one per object). According to the codebook entry each matched feature has several corresponding 3D model locations. To robustly estimate the 6D object pose we use the OpenCV pose estimation procedure in a RANSAC [27] scheme with a probabilistic termination criterion, where the number of iterations necessary to achieve a desired detection probability is derived from an estimate of the inlier ratio, which is taken to be the inlier ratio of the best hypothesis so far. So the number of RANSAC iterations is adapted to the difficulty of the current situation and accordingly easy cases quickly converge.

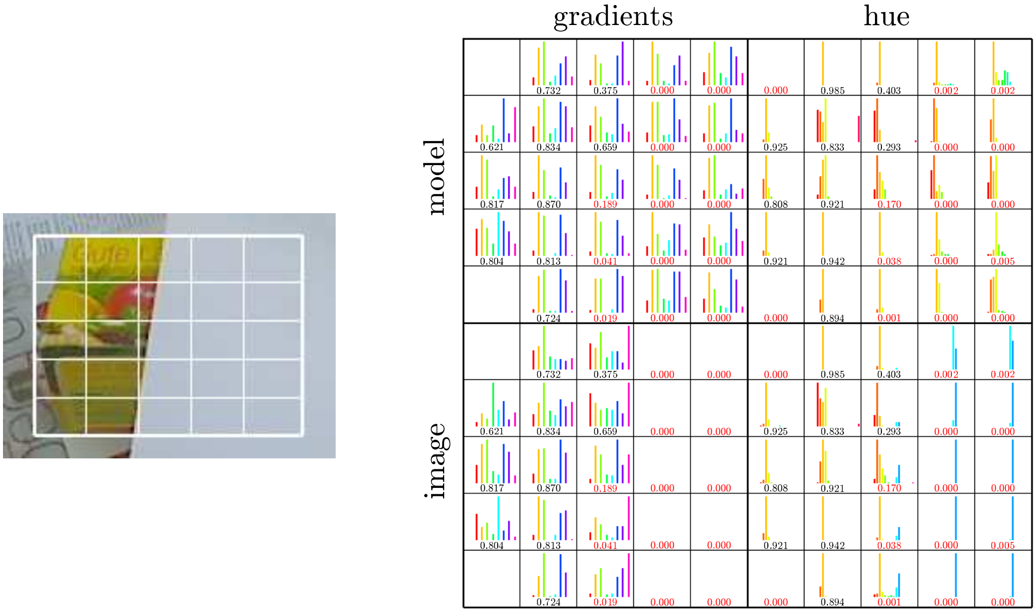


Figure 3.9: Histogram descriptor for an occluded object. The intersection values of the gradients- and hue histograms approximate the amount and location of the occlusion.

3.3.4 Model completeness

Now that it is possible to learn texture and SIFT based features of the model, the question arises when to stop learning. In other words, how much information is needed to represent the model sufficiently for tracking, initialisation and recovery of the pose. For tracking, completeness is achieved if the textures of all faces of the model are captured according to Section 3.3.2. Unfortunately this can not be applied to the SIFT based model since detection suffers much more from angular deviation and scale. Therefore [105] proposes a view-based probabilistic formulation indicating how likely it is to detect the object from a certain point of view. In more detail, the probability of detection success $o_t \in \{true, false\}$, given the object pose $\mathbf{x}$ is formulated using Bayes rule.

$$p(o_t|\mathbf{x}) = \frac{p(\mathbf{x}|o_t)p(o_t)}{p(\mathbf{x}_t)} = \frac{p(\mathbf{x}|o_t)p(o_t)}{\sum_{k\in\{true,false\}} p(\mathbf{x}_t|o_t = k)} \tag{3.17}$$

The probability $p(\mathbf{x}|o_t)$, i.e. of observing a particular pose $\mathbf{x}$ for a detected or missed object view o_t is estimated from labelled training data. This data is obtained by transforming a virtual object model with 1000 random rotations, 252 scales and varying levels of artificial noise and blur. The prior $p(o_t)$ is the probability of detecting the object at all, which might come from contextual information, e.g. the probability of an object being in a certain room. For our experiments $p(o_t)$ is set to 1. To come to a measure of *model completeness* the probability of detection over all learned views is taken.

$$\hat{p}(o) = \sum_{\mathbf{x},t} p(o_t|\mathbf{x})p(\mathbf{x}) \tag{3.18}$$

where $p(\mathbf{x})$ takes into account that certain views are less likely than others (such as the underside of an object). This representation allows a robotic system to identify lack of information and to take action to learn more views (e.g. repositioning, moving the object or the camera, etc.). E.g. in the work of [105] a gain-cost-scheme is applied to drive exploration.

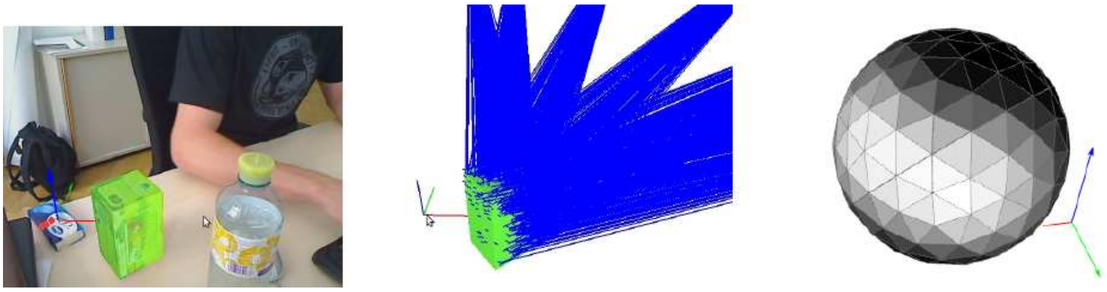


Figure 3.10: Model completeness: The object in the scene (left) and bundles of features with their view vectors (middle) after acquiring some views of the object. View sphere (right) with brighter shades of grey indicating that the object has been learned from the respective direction.

In our approach the object poses relative to the camera are represented by the unit sphere, disregarding the distance in between. Figure 3.10 shows such a sphere, where bright regions indicate viewing angles of high probabilities whereas dark ones have not been learned so far.

3.4 Results

We evaluated the approach by using virtual rendered image sequences with known ground truth as well as live sequences where we obtain ground truth from a calibration pattern rigidly attached to the object. All experiments were performed on a PC with an Intel Core 2 Quad (Q6600, 2.4 GHz) CPU, a NVIDIA GeForce GTX 285 GPU and a Logitech Webcam Pro 9000 run at a resolution of 640x480 pixels.

3.4.1 Evaluation of the tracking error

For a measure of the error we used the scheme proposed in Section IV-B in [47], where a large number, $k = [1 \dots K]$, of randomly chosen points $\mathbf{q}^k \in \mathbb{R}^3$ are rigidly attached to the object surface at the ground-truth pose and compared to the corresponding points $\hat{\mathbf{q}}^k \in \mathbb{R}^3$ of the tracked pose. The error at a specific frame t is then approximated by

$$e_t = \frac{1}{K} \sum_{k=1}^{K} |\hat{\mathbf{q}}_t^k - \mathbf{q}_t^k| \tag{3.19}$$

Before evaluating our method in terms of the above error metric, let us briefly consider the possible sources of errors in our system, such as errors from calibration, geometric modelling, image quantisation and finally the tracking algorithm itself. Concretely we identify the following sources of errors:

- *Mechanical error:* Positioning the calibration pattern rigidly on the object introduces a small unknown error which can safely be considered to be in submillimetre range.

- *Camera error:* The pose of the calibration pattern is detected with a standard DLT algorithm, followed by a non-linear optimisation of the pose using the sparse bundle adjustment implementation by Lourakis [59]. Further the rolling shutter of the camera used introduces additional errors, which is negligible for our speeds.

- *Quantisation error:* Depending on image resolution a digital camera introduces a pixel quantisation error. In our evaluation we use a resolution of 640x480 with a focal length of ∼500 in pixel-related units. This leads to an error of about 0.5-1.5 mm when tracking at a distance of 0.5-1.5 m parallel to the image plane. This error is even higher for the orthogonal direction, which shows up in Table 3.1.

- *Modelling error:* For modelling we measured the main dimensions of the objects used, but we used simplified models that do not account for deviations like small details, chamfers or slightly bulging cardboard surfaces. Unfortunately we do not have a measure for the *Modelling error* but since we mainly used basic shapes, where correct modelling is simple, we assume this error to be negligible.

- *Texturing error:* We found that textures added during the modelling phase do not align properly. Manually capturing textures triggered by pressing a button incorporates less error than automatic capturing based on tracking-state-detection.

- *Tracking error:* The failure of the tracker to accurately locate the local maximum, depending on the challenges posed by current viewing conditions.

3.4.2 Accuracy and precision

Accuracy is defined to be the closeness of a quantity to its actual value, which in our case is measured using Equation (3.19), where the pose of tracking is compared to the pose of the virtual object and the pose detected by the calibration software respectively. We evaluated the mean accuracy with respect to the poses of several trajectories using

$$e = \frac{1}{Jt_e} \sum_{j=1}^{J} \sum_{t=1}^{t_e} e_t \tag{3.20}$$

where $j = [1 \dots J]$ are the trajectories of poses $t = [1 \dots t_e]$ under unchanged conditions, i.e. tracking J times on a sequence of t_e images.

Precision, also called repeatability, is the degree of deviation of a quantity under unchanged conditions, which is measured using Equation (3.19). For each frame t, the pose of tracking $\hat{\mathbf{q}}^k$ is compared to its own mean with respect to the number of repetitions J. I.e. the points of ground-truth $\mathbf{q}^k$ in Equation (3.19) are substituted by

$$\mathbf{q}_t^k = \frac{1}{J} \sum_{j=1}^{J} \hat{\mathbf{q}}_t^k \tag{3.21}$$

and precision is again given by Equation (3.20).

Table 3.1: Accuracy and Precision

Target Object	Accuracy [mm]				Precision [mm]			
	static		dynamic		static		dynamic	
	x,y	z	x,y	z	x,y	z	x,y	z
box (virt.)	0.4	2.3	1.5	5.6	0.2	1.1	0.7	3.2
box (real)	2.0	5.5	2.6	7.7	1.1	2.9	1.6	4.9
cylinder (virt.)	0.9	4.4	2.4	10.0	0.4	1.9	1.3	5.7
cylinder (real)	3.0	16.5	3.9	21.9	0.5	2.5	1.6	8.8

Table 3.1 shows the results of the accuracy and precision evaluation, where we evaluated two different cases: A *static* scene where we looked at the mean error of the pose after it converged within a few frames. And a *dynamic* scene where we

observed the mean error of the trajectories. For evaluation we used box shaped and cylindrical objects. The virtual objects show the *Tracking error* and *Quantisation error* (all other errors being ruled out), whereas the difference between virtual and real objects are due to *Mechanical*, *Camera*, *Modelling* and *Texture error*, where we assume the *Modelling* and *Texture error* to play the main roles. We evaluated the dynamic errors under the following conditions:

- Linearly moving objects with different velocities.
- Rotating objects.
- Moving objects arbitrarily (i.e. toppling, rolling).
- Partially occluded objects.
- Changing illumination.

We can derive from Table 3.1 that curved objects are typically harder to track than box-shaped objects. A typical trajectory for arbitrary movement is shown in Figure 3.11 where the tracked pose is compared to the virtual with respect to translations, rotations and the error measured by Equation (3.19). For generating the virtual poses the poses from the pattern detection and bundle adjustment were used and filtered to remove jitter.

Pose alignment: Table 3.1 indicates that the tracking accuracy is significantly better in x- and y-direction than in the direction of the camera view ray (z-direction). This results from the projective nature of the camera. For reconstruction as described in Section 2.5 the accuracy given would result in bad object models. To account for this deficiency the current pose of the object is corrected. Therefore the object shape is projected into the image space of the new key-frame assigning a depth value to each pixel. These depth values are compared with the depth of the range image captured by the camera, within the region of overlap. The mean of the differences of the depth values (signed error) is used to correct the pose with respect to the z-direction in the current key frame.

3.4.3 Robustness

We tested our approach against various situations including

- fast movement introducing motion blur,
- occlusion,
- changes in lighting,
- large distances, small objects,
- different objects (high resolution, curved surfaces, low texture, . . .).

Since robustness is hard to put in numbers the reader is referred to the video described in Section 3.4.5, to get an impression of how these various challenges are handled.

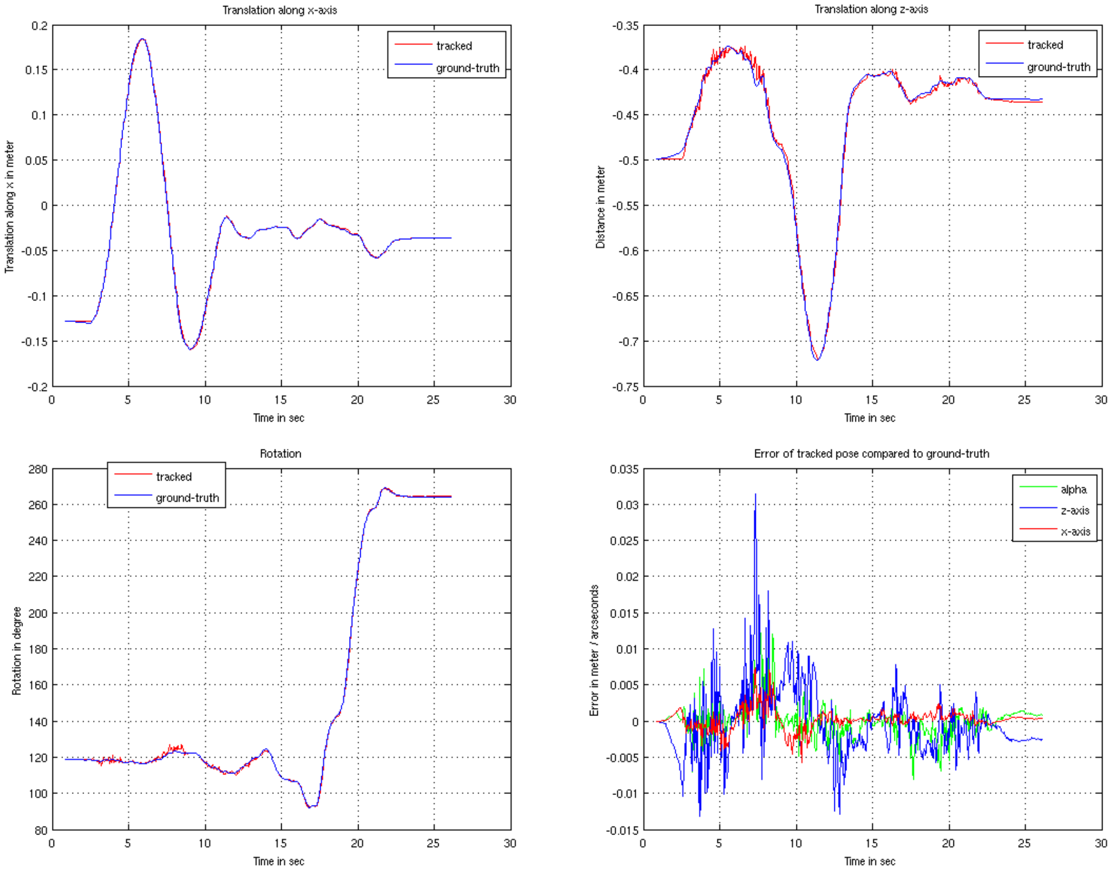


Figure 3.11: Trajectory of a tracked virtual object with 45 cm x-translation followed by a 70 cm z-translation and a rotation about the objects y-axis. The lower right figure shows the pose deviations respectively.

3.4.4 Performance

Processing time during tracking depends, on the complexity of the model as well as on the number of particles used for tracking.

Table 3.2 shows the frame rates for different numbers of faces and particles. 2x50, 3x100 and 4x300 indicates 2, 3 and 4 iterations using 50, 100 and 300 particles for each iteration respectively. Figure 3.12 shows the frames per second on different GPUs with respect to the total number of particles used for tracking.

3.4.5 Video

The video[1] shows how we learn texture and feature points during tracking. Then TSD identifies whether an object is occluded or tracking fails, in which case pose recovery is triggered automatically. Since pose recovery also takes advantage of GPU computing the tracker slows down at this particular moments. Note that we do

---

[1] Video: `http://users.acin.tuwien.ac.at/tmoerwald/?site=4` (bottom)

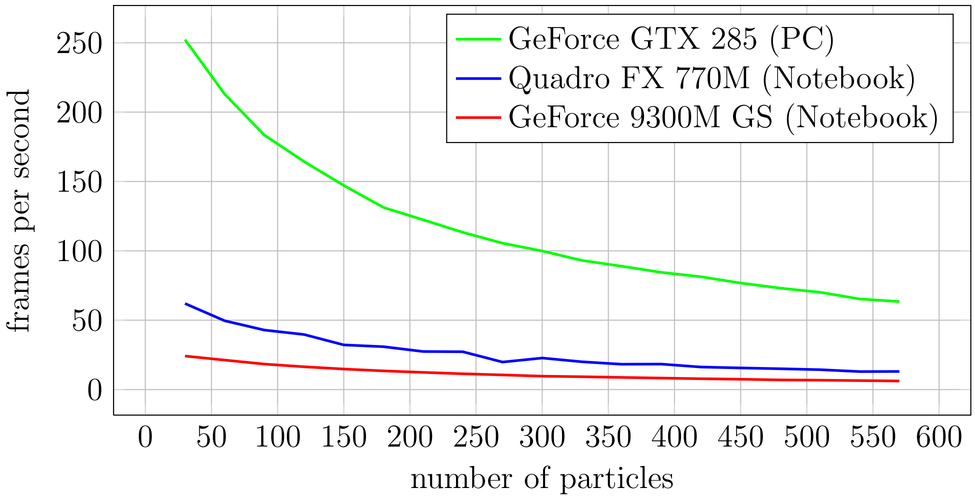


Figure 3.12: Frame rates with respect to the number of particles on different platforms for the *Box* model.

Table 3.2: Frame rates with respect to model complexity and number of particles

Example Objects	Faces	Frames per Second		
		2x50	3x100	4x300
Box	6	240	100	33
Cylinder (low)	24	220	95	30
Cylinder (mid)	96	210	90	28
Cylinder (high)	384	190	80	25

not interfere with the tracking system via the keyboard other than for changing the visualisation modes.

3.5 Discussion

In this chapter we described how the pose of the object of interest is tracked robustly. We have modified and improved state-of-the-art particle filtering approaches by various contributions. Defining the variance of the particles depending on their confidence from the previous observation yields faster convergence and less jitter. Fixing the pose of some of the best particles further reduces jitter as no good poses are lost due to re-sampling. Further this fixed poses indicate the tracking state. Another method improving visual tracking based on particle filtering is our iterative structure leading to a faster convergence by sampling fewer particles more often.

Further we have developed a method to determine the state of tracking in a qualitative way. This allows us to reason about the quality of a certain trajectory and to identify good views. The first is needed in the following chapter, where the physical behaviour is learned by visual tracking, only taking into account trajectories of a certain quality. The latter was already applied twice. First, when learning the shape of an object from multiple views as in Section 2.5 and second when learning the texture and colour features which are then mapped onto the surface in Section 3.3.2 and 3.3.3.

In our scenario of a cognitive robot it is necessary to drive the robot's interest in learning objects. Considering the requirements of a cognitive task execution the robot therefore needs to reflect its current knowledge. In the case of object tracking this reflecting is modelled by a probabilistic representation, namely the view sphere. It allows to add already learned regions and search for views from which the object has not been observed yet.

A not so obvious but necessary requirement for TSD is detection of occluded objects as a special case of a tracking state. Its importance becomes clear when considering that we start learning from scratch, and we do not want to improve our models by adding information of other occluding objects. This means that we have to detect views where such a situation occurs and mark them as being not good to learn from.

Note that this chapter provides the tools for our cognitive robot, rather than describing new cognitive methods. Therein also lies the main contribution to the community. Although a lot of tracking algorithms exists, there are hardly any that allow for robust tracking and provide qualitative statements about the observations. In other words our visual tracking method with its TSD corresponds to the reflectance block in Figure 1.2 executed in real-time.

Chapter 4

Physical prediction and robotic manipulation

This chapter describes how physical behaviour of objects can be learned in a cognitive sense. That is, to allow for modification and extension as new information about the object arrives. As already mentioned in Chapter 1, classical mechanics based on Newtons laws are not appropriate for three reasons. First, they strongly depend on the shape and completeness of the object model. Consequently the parameters learnt are worthless once the shape changes. Second, they are only idealized models of reality and third, even if the are capable of making good predictions, it might be difficult to estimate all parameters required for a particular case. Although physical model generalise well to changing parameters, in the sense that predictions look qualitatively reasonable, their predictions may be far from the actual ones.

A more reasonable representation is to use probabilities assigned to certain movement configurations. This implicitly encodes basic physical effects such as impenetrability, gravity and inertia as well as more complex ones as friction and dynamic movement. The model is realised by different *experts* which are attached to global or local coordinate frames. *Locally weighted projection regression* (LWPR) and *kernel density estimation* (KDE) are employed to predict the movement of objects, where KDE allows to generalize the learnt probabilities to different configurations and shapes. Further we show how this representation can be used to improve robustness of visual tracking for a robotic system. The physical knowledge allows to handle difficult situations like full occlusion and motion blur due to fast movement.

In our scenario a robot arm, equipped with a single rigid finger, applies a series of pokes or pushes to the object, causing it to move from one pose to another. Doing so, the system observes the object using the visual tracking algorithm described in Chapter 3 and thereby builds up its physical knowledge. After several trials the robot is able to predict the movement of the object which is used to constrain tracking. By detecting visual distraction with the tracking-state-detection (TSD), the predictor steps in during phases of heavy distractions, guiding the tracker to physically feasible poses. Finally these poses are validated by the Monte Carlo particle filter, described in Section 3.2.2, where again the most probable ones are fed back to the prediction experts. TSD allows for further training of the experts leading to better predictions and consequently better tracking. This demonstrates the usability of detecting gaps in the knowledge representation of a cognitive robot.

It may seem somewhat esoteric to focus on pushing, however there are good reasons to do so. Pushing is a very fundamental form of manipulation. More complex activities, such as dexterous in-hand manipulation with multiple fingers, can be viewed as combinations of many simultaneous single-finger actions. More practically, even industrial pick and place operations with a simple two jawed gripper often result in a pushing phase, where uncertainties in both object and robot pose lead to one jaw contacting the object before the other. This can even lead to gross grasp failures when the object topples over under pushing from the first jaw, before the second jaw makes contact. Therefore it is important to solve the problems of robotic pushing, and controlled pushing will typically be reliant on tracking the object pose with a vision system.

Overview: In this chapter, we provide an overview of the "learning-to-predict" architecture, and then show how it can be conveniently incorporated into a particle filter-based vision algorithm to propagate particles from one frame to the next. We demonstrate the effectiveness of the technique, for tracking pushed objects past large occlusions and other difficult circumstances, where attempting vision without adequate prediction would fail.

We start by giving an overview of related work on physical motion models for objects under robotic manipulation and their application for visual tracking in Section 4.1. Section 4.2 provides an overview of our system for learning to predict the outcomes of manipulative pushes. We describe how the motions of rigid bodies are represented by coordinate frames and transformations. We show how objects and their motions can be decomposed and how a variety of probabilistic *experts* can be trained to predict various aspects of these motions. We show how to combine the opinions of these experts as a product of densities, which is capable of significant generalization to new objects with different shapes and different push directions which have not been encountered during training. In Section 4.3 we extend our algorithm for visual tracking of 3D objects [75, 72, 71, 74] with the prediction framework introduced in Section 4.2. We show how tracking can be improved by incorporating a well trained predictor. Section 4.5 presents results of this work, providing examples of how the enhanced tracker copes with difficult situations such as occlusion and motion blur. Figure 4.1 gives a high-level overview of our tracking and prediction framework.

4.1 Related work

There is a limited body of literature describing vision algorithms tailored specifically for robotic manipulation tasks. For example, Drummond and Cipolla [23] incorporate knowledge of kinematic constraints into tracking of articulated chains of rigid bodies,

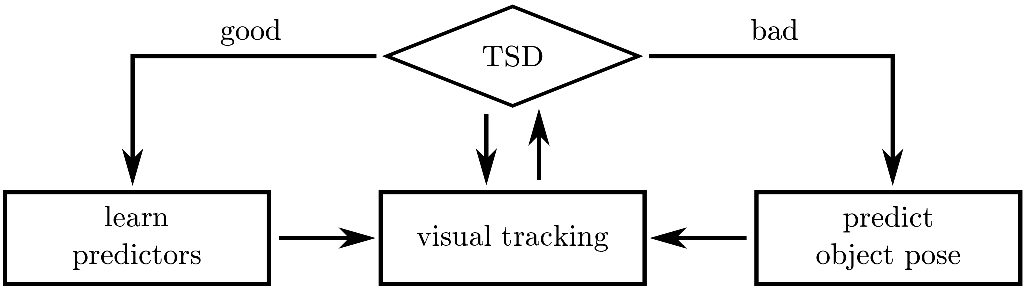


Figure 4.1: Simultaneous tracking and predicting the object pose.

with a view to tracking robotic arms for visual servoing. However, it is more usual for researchers to simply take a generic tracking algorithm and incorporate it with an existing manipulation planning system, e.g. [50]. Typically the vision algorithms are drawn from the model based tracking literature, for example [36, 60, 102], which predominantly track by choosing candidate poses of the tracked body, whose projected wire-frame edges best match edges extracted from images. More recently, the ability to make use of advanced graphics cards for high speed projective calculations, means that such techniques can be applied to tracking with robust particle filters, e.g. [42, 9, 76].

Particle filters rely on motion models, to propagate particles from one predictor-corrector step to the next. In practice, little may be known about the motion of the tracked object, and so predominantly these motion models must be very blunt instruments. It is typical to simply apply Gaussian noise to particles to propagate them, assuming no real understanding of how the object might move at the next time step. However, if the tracked object is subject to robotic manipulation, we should be able to make use of our knowledge of the planned manipulative action, to make a much more informed prediction of the next phase of the objects motion. In the case of an object which is rigidly held in the jaws of a hand or gripper, the motion prediction problem becomes trivial since the object is exactly constrained to follow the motion of the manipulating arm. However, in pushing manipulation, the motion of an object which will result from an applied single-finger push or poke is much more uncertain.

Early approaches to predicting the effects of robotic pushes on object motion, [65, 81, 62, 1, 63], attempted analytical solutions of physical constraints. These approaches did not progress beyond anything more complex than the simple 2D case, with flat polygonal objects, constrained to slide on a planar surface. More recently, Cappelleri [8] used physics simulation software to plan manipulative pushes, but again this was limited to a 2D problem, with a small, flat rectangular object which was constrained to slide while floating on a film of oil to simplify frictional interactions. We know of little in the way of literature which specifically addresses the prediction problem in robotic push manipulations of real 3D objects, which are subject to complex 6-dof motions such as tipping and toppling over. It is possible to use physics simulators to predict the motions of interacting rigid bodies, however this approach is reliant on explicit knowledge of the objects, the environment and key physical parameters which can be surprisingly difficult to effectively tune in practice, [24]. Furthermore, once a physics simulator has been set up for a particular scenario, it is not generalizable to new objects or novel situations.

In contrast, recent works by Kopicki et al. [45, 47, 46, 48] propose a system which can learn to predict the explicit 3D rigid body transformations. The system does not make use of any physics simulation, or any hard coding of Newtonian physics equations or physical constraints. Instead, a statistical relationship between applied pushes and resulting object motions is trained, by simply having the robot apply a series of random pushes to the object, recording the finger trajectories, and observing the resulting object motions with a vision system.

The work in this chapter was achieved together with my colleague Marek Kopicki from the University of Birmingham. Work on predicting and learning physical movement is still going on and results achieved so far are quite promising especially for unknown objects and environments.

4.2 Prediction

This section is just a brief overview of the work in [46, 45] to show how rigid body movement can be described in a probabilistic form.

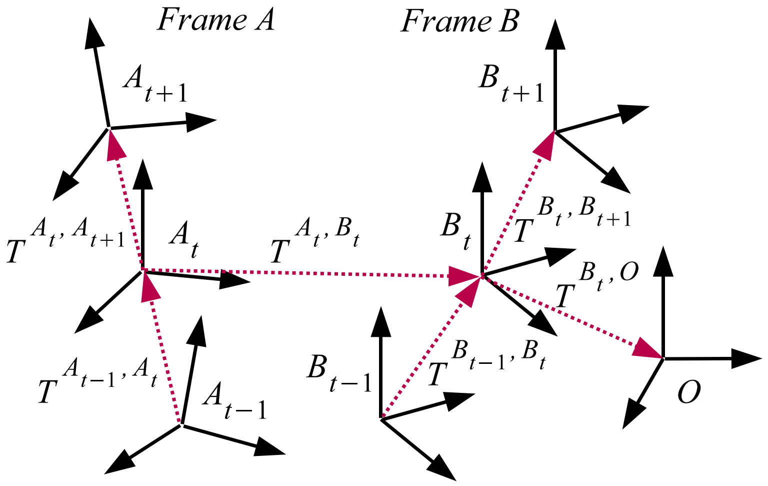


Figure 4.2: A system consisting of three interacting bodies with frames A and B and some constant environment with frame O.

A system consisting of three interacting rigid bodies can be described by coordinate frames A, B and O and by six transformations between the bodies and different time steps ($t-1$, t and $t+1$), with respect to a constant environment O as shown in Figure 4.2. A and B change in time and are observed at discrete time steps $\ldots, t-1, t, t+1, \ldots$ every non-zero Δt. As stated in [46] a triple of transformations $\mathbf{T}^{A_t,O}$, $\mathbf{T}^{A_{t-1},A_t}$ and $\mathbf{T}^{A_t,A_{t+1}}$ provide a complete description of a state of a rigid body A in terms of classical mechanics. Of course the same is true for some body B. The prediction problem can be stated as: given we know or observe the starting states and the motion of the pusher, $\mathbf{T}^{A_t,A_{t+1}}$, predict the resulting motion of the object, $\mathbf{T}^{B_t,B_{t+1}}$. This is a problem of finding a function:

$$f : \mathbf{T}^{A_t,B_t}, \mathbf{T}^{B_t,O}, \mathbf{T}^{A_{t-1},A_t}, \mathbf{T}^{B_{t-1},B_t}, \mathbf{T}^{A_t,A_{t+1}} \rightarrow \mathbf{T}^{B_t,B_{t+1}} \tag{4.1}$$

In many robotic applications manipulations are slow, so we can assume quasi-static conditions and it is often possible to ignore all frames at time $t-1$. This conveniently reduces the dimensionality of the problem, giving:

$$f : \mathbf{T}^{A_t,B_t}, \mathbf{T}^{B_t,O}, \mathbf{T}^{A_t,A_{t+1}} \rightarrow \mathbf{T}^{B_t,B_{t+1}} \tag{4.2}$$

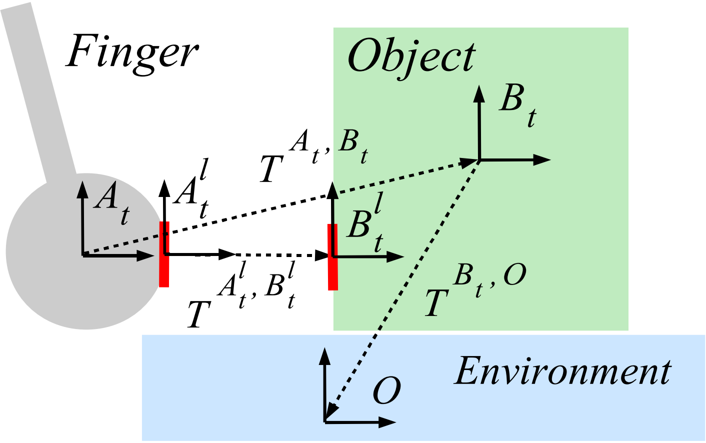


Figure 4.3: A robotic finger A pushing an object B in an environment O, decomposed into local and global coordinate frames.

Prediction learning using Functions (4.1) or (4.2) is limited with respect to changes in shape (see Chapter 5.3 of [45]). The problem can be expressed by a product of several probability densities over the rigid body transformation, encoding global as well as local contact configurations. Figure 4.3 shows the frames representing two different *experts* [45].

$$\begin{aligned} p_{\text{global}} &\equiv p_{\text{global}}(\mathbf{T}^{B_t,B_{t+1}} | \mathbf{T}^{A_t,A_{t+1}}, \mathbf{T}^{A_t,B_t}, \mathbf{T}^{B_t,O}) \\ p_{\text{local}} &\equiv p_{\text{local}}(\mathbf{T}^{B_t^l,B_{t+1}^l} | \mathbf{T}^{A_t^l,A_{t+1}^l}, \mathbf{T}^{A_t^l,B_t^l}) \end{aligned} \tag{4.3}$$

In addition to learning how an object moves in response to a push, it is desirable if we can also incorporate learned information about the inherent tendencies of parts of an object to move in various directions with respect to the environment or any other objects, but regardless of whether it is being pushed or not. This additional information may help when predicting the motions of previously unseen objects, because it provides some prior knowledge about what kinds of motions are possible and which are not. The subsequent motion of the object in the inertial frame can now be described as:

$$\begin{aligned} p(\mathbf{T}^{B_t,B_{t+1}}|K) &= \\ p_{\text{local}}(\mathbf{T}^{B_t,B_{t+1}}|\mathbf{T}^{A^l_t,A^l_{t+1}}, \mathbf{T}^{A^l_t,B^l_t}) &\times \\ p_{\text{global}}(\mathbf{T}^{B_t,B_{t+1}}|\mathbf{T}^{A_t,A_{t+1}}, \mathbf{T}^{A_t,B_t}, \mathbf{T}^{B_t,O}) & \end{aligned} \tag{4.4}$$

where K stands for the known conditions $\mathbf{T}^{A^l_t,A^l_{t+1}}$, $\mathbf{T}^{A^l_t,B^l_t}$, $\mathbf{T}^{A_t,A_{t+1}}$, $\mathbf{T}^{A_t,B_t}$ and $\mathbf{T}^{B_t,O}$. The product of probability densities allows to incorporate even more information, like the local shape densities with respect to the environment as in [48].

Implementation We presented two formulations of the prediction problem: 1) as function of approximation in Equation (4.2), and 2) as density estimation in Equation (4.4). Further we have suggested that there may be an advantage of solving the density problem by applying the product of experts in Equation (4.4).

- *Regression method:* We used *locally weighted projection regression* (LWPR) [96] to estimate the mapping described by Equation (4.2). The regression scheme was implemented using the LWPR software library [41].

- *Single expert and multiple expert methods:* A variant of *kernel density estimation* (KDE) is used to approximate conditional densities as

  $$\widetilde{\mathbf{T}}^{B_t,B_{t+1}} = \underset{\mathbf{T}^{B_t,B_{t+1}}}{\arg\max} \{p_{global} p_{local}\} \tag{4.5}$$

  The density product is maximised using the differential evolution optimisation algorithm [93].

Rigid body transformations used in both were parametrised as described in Section 3.2.1. For more details see [48, 46].

4.3 Tracking

For visual observation of the trajectory of the object we use the tracker introduced in Chapter 3. To obtain better results for tracking we conveniently fuse prediction into the particle filtering framework. Inserting Equation (4.4) into (3.6) and substituting $\widetilde{\mathbf{T}}^{B_t,B_{t+1}}$ with $\mathbf{x}^{B_t,B_{t+1}}$ according to Equation (3.1) yields to

$$p(\mathbf{x}^i_t|\mathbf{x}^i_{t-1}, c_{t-1}) \sim \mathcal{N}(\mathbf{x}^i_{t-1} + \mathbf{x}^{B^i_{t-1},B^i_t}, \boldsymbol{\sigma}^2_{t-1}). \tag{4.6}$$

Therefore the prediction $\mathbf{x}^{B_t,B_{t+1}}$, with $\mathbf{x}^{B^i_t,O^i_t} = -\mathbf{x}^i_t$, is only calculated when a new image arrives from the image capturing pipeline, otherwise it is set to 0. After the prediction step t is incremented by 1. Figure 4.4 shows the modified tracking algorithm.

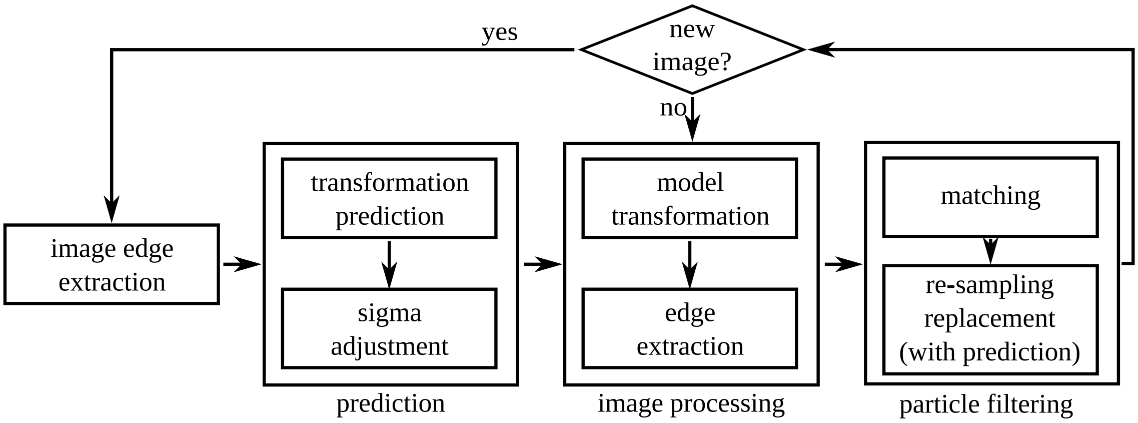


Figure 4.4: Tracking with motion prediction within the iterative particle filtering framework.

The particle filter always tries to find the local maximum in observation space. In the case of occlusion, as shown in Figure 4.7 and 4.9, this leads to drifting of the tracker away from the real pose of the object. To cope with this problem we employ the tracking-state-detection explained in Section 3.3.1 and turn off noise $\mathcal{N}$ whenever the tracking quality is too low, i.e. $\boldsymbol{\sigma}_{t-1}^2$ is set to 0.

$$\boldsymbol{\sigma}_{t-1}^2 = 0 \qquad\qquad \text{if quality} = \text{bad} \tag{4.7}$$

With respect to Equation (4.6), this means that the system completely relies on the output of the predictor.

4.4 Learning

The prediction system can be trained by observing the outcomes of random robotic pushes, and extracting the resulting object motions with the tracker presented in Chapter 3. For the first trials the tracker does not take into account the prediction. This decreases the robustness of the tracker with respect to occlusion or motion blur. However, in a tutor-driven learning scenario these disturbances can be limited. Furthermore the tracking-state-detection allows to differentiate between good and bad trajectories, where the latter are simply neglected. In return, the physical prediction system provides prior poses for the tracker, increasing accuracy and robustness. Furthermore situations of full occlusion or strong motion blur may be overcome using these predictions as will be shown in Section 4.5.

4.5 Results

The prediction system used is evaluated with respect to the tracking system. The multiple experts and regression methods are evaluated against each other and to prediction using Newtonian mechanics. Further the generalisation of the proposed representation to novel pushing configurations and object shapes is tested. In all the experiments we used simple object shapes like boxes, cylinders and polyflaps, which are two planar, rectangular plates rigidly connected on one of each edge. Polyflaps were chosen since they provide a wide range of interesting movements (sliding, toppling, rotating and tilting while not toppling).

4.5.1 Tracking

In all our experiments we are using the tracking system as described in Chapter 3, and compare it against the tracker without prediction which is the same system but without the *motion prediction* step as in Figure 4.4. Other than *motion prediction* we are using the same configuration for each of the trackers, respectively non-iterative particle filtering with 100 particles for each frame. The number of particles was chosen small enough to ensure real-time operation in normal conditions, which however meant the tracker would run into problems as conditions deteriorate. One option in such a case would be to increase the number of particles, accepting loss of real-time performance (and e.g. buffering images), and indeed the tracker allows such dynamic resizing of the particle set. The approach is to rely on an improved motion model based on the learned predictors rather than throw more particles at the problem, which allows us to also cover very severe cases where no number of particles can maintain a successful track. The following experiments are designed to illustrate the differences in performance between tracking with and without incorporating a learned prediction system. The poses are drawn as wire-frame models with the following colour-code:

- *White*: Ground truth. Note that we did not use an external system such as a magnetic tracker for obtaining ground truth, but used the visual tracker itself in a high accuracy non real-time setting with many iterations and particles (4 and 200 respectively).
- *Green*: Tracking with prediction.
- *Red/Magenta*: Tracking without motion prediction.
- *Blue*: Pure motion prediction without visual feedback by the tracker, i.e. in Equation (4.7) $\mathbf{n}_{t+1} = 0$.

For visual observation a camera is capturing images with a resolution of 800×600 at a frame rate of 30 Hz and highly accurate time-stamps. Tracking is executed in real-time whereas in critical situations, where the prediction system has to take over, the data is buffered and evaluated at a frame rate of 1-5 Hz.

Tracking accuracy

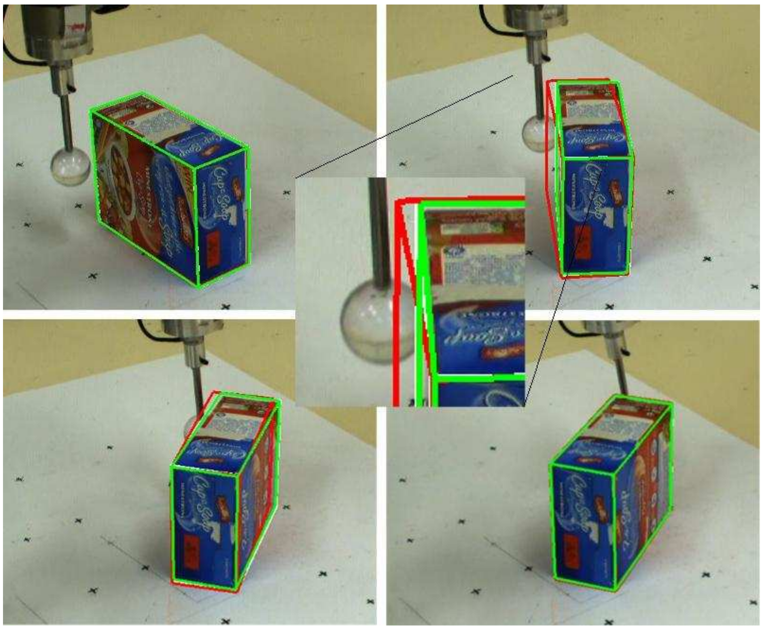


Figure 4.5: Accuracy: Overlaid edge-based tracking results with and without prediction, from top-left to bottom-right. (ground truth: white, tracking with prediction: green, tracking without prediction: red)

In this experiment we show how accuracy increases using the proposed methods. We compare the poses of tracking with and without prediction against ground truth. To evaluate the error in both cases we used the non-normalized error measure described in Section IV-B of [47].

$$E_t = \sum_{n=1 \ldots N} |p_n^{2,t} - p_n^{1,t}| \tag{4.8}$$

where $p_n^{1,t}$ are randomly chosen points on the object surface at the ground truth pose. Thus E_t measures the mean displacement of the object surface.

For a fair comparison we only used pushing examples, where also the tracker without prediction was able to maintain tracking throughout the whole sequence, as shown in Figure 4.5. In the upper-left image both trackers are initialised at the same pose. The upper-right and lower-left show the mismatch of the tracker without prediction (*red*) while the object is moving. At the end of the sequence, lower-right image, both trackers converge to the ground-truth pose as to be expected.

Figure 4.6 shows the result of the evaluation of 20 pushes. For both, average error and standard deviation, the tracker which takes advantage of information from the predictor performs significantly better.

The following experiments show robustness to various events in a qualitative manner. For these cases a quantitative evaluation against the tracker without prediction is meaningless, as the latter loses the object at some point altogeher.

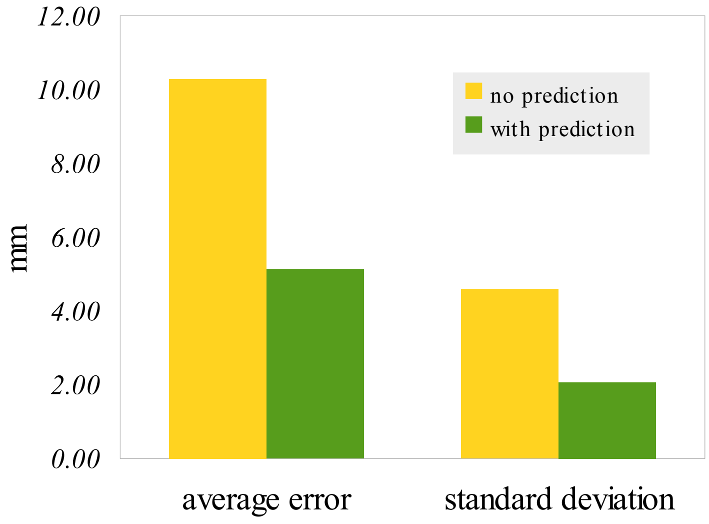


Figure 4.6: Tracking accuracy with and without prediction: average displacement error of surface points.

Robustness – occlusion

Typically visual tracking algorithms fail when the object is partially or completely occluded. The prediction model of Section 4.2 allows us to overcome such situations. Figure 4.7 shows a pushing sequence which suffers from heavy occlusion. At the beginning both of the trackers perform very well, since at least parts of the object are visible (top row). At the point where the occluder is completely hiding the object, the tracker without predictor (magenta wireframe) is drifting away to a visually more likely position (e.g. it is attracted to the robot hand which introduces clutter with respect to edges as well as colour) and fails to keep track of the object pose.

Pure prediction (blue wireframe) does not use any visual feedback and produces the whole trajectory from the initial pose. The finger is pushing very near to the centre of the object which is a very unstable position. Given this push, the object in some cases might slide to the left or to the right, or topple over depending on slightly different initial positions. However, since the predictor used for tracking gets updated by the visual observation it is possible to handle such difficult situations.

Robustness – motion blur

Another example of a difficult situation is fast movement of the object relative to the camera. Figure 4.8 illustrates such a case, where a box is pushed forward causing it to tilt until it reaches an unstable pose and finally toppling over. This is a very critical situation for visual observation. The falling object moves quite fast, causing the image to blur.

Again the tracker with (green) and without prediction (red), and the pure predictor (blue) are initialised at the same starting pose. During the first phase of the

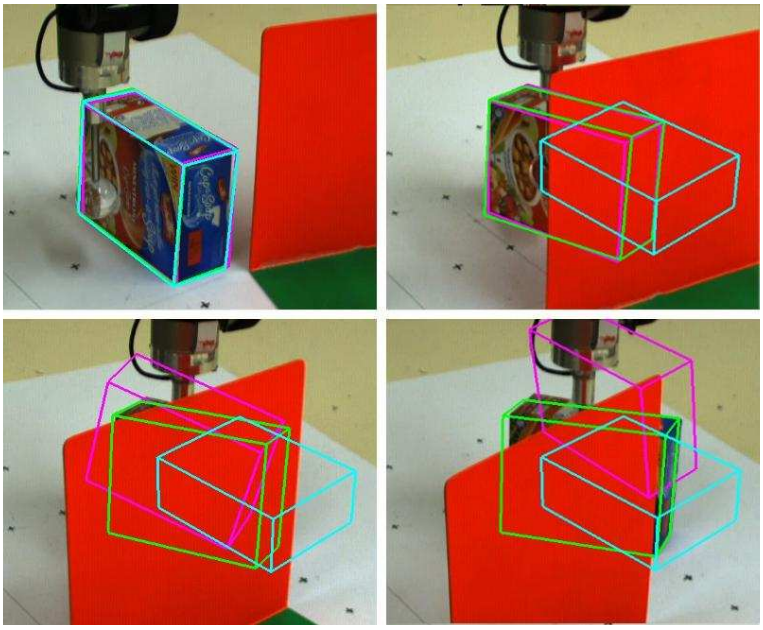


Figure 4.7: Failure of tracking without prediction in case of occlusion. (edge-based tracking with prediction: green, without prediction: magenta, pure prediction: blue)

sequence the predictor proposes an erroneous rotation of the object, while the vision system extracts the correct pose relatively accurately (top row of Figure 4.8). However, by the time of the unstable pose shown in the lower-left image the tracker with prediction is already better than the tracker without prediction. The object is moving fast during the next frames causing the effects mentioned above. The tracker without prediction can not follow the fast movement, loses track and gets trapped in a local maximum. The predictor on the other hand proposes the right pose and the corresponding tracker refines the result.

Robustness – occlusion with motion blur

The hardest case for a visual observation system is the combination of fast movement and occlusion. We tested this case by applying a pushing manipulation where the object is hidden behind an occluder where it topples over, as shown in Figure 4.9.

In the top-left image enough parts of the object are visible and both of the trackers produce good results. The top-right image shows the object behind the occluder already in the phase of falling down as the blur suggests. The lower-left is the subsequent frame and illustrates the large change of the pose, which causes the tracker without prediction to fail, whereas the tracker with prediction overcomes this difficult situation. The pose of the pure predictor is also very close to the real one, but suffers from integrating error over the trajectory.

Note that for this experiment we placed a virtual occluding object in the scene. This allowed us to vary the size and texture of the occluder and most importantly to position it right in front of the toppling object.

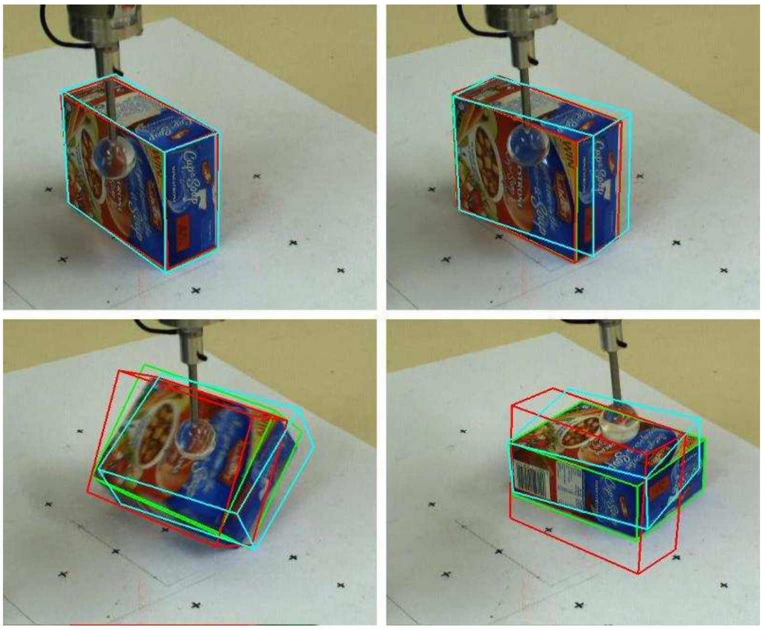


Figure 4.8: Failure of tracking without prediction in case of a toppling object. (edge-based tracking with prediction: green, without prediction: red, pure prediction: blue)

4.5.2 Prediction

For evaluating the prediction we use a 5-axis Katana robotic manipulator equipped with a single rigid finger. The motion of pushed objects is captured using a single camera and the tracking system described previously. Note that the tracking system was only used for learning and not for testing, since we want to evaluate purely the predictors. Simulation experiments are carried out using the NVIDIA PhysX physics engine, providing perfect ground-truth data. The parameters of the physics engine are automatically tuned to best fit the world. As expected we found that the parameters neither correspond to their true values, nor do they generalise.

Comparison with prediction using Newtonian mechanics

We obtained predictions using the NVIDIA PhysX physics simulator, with estimated parameters that match the real data the best. Figure 4.10 presents a comparison of the physics simulation and the learned predictors (trained on 900 trials). Figure 4.11 shows an example of a polyflap being pushed by a robotic finger. Clearly the physics simulator is unable to match predictors trained in a real experiment, even though the real training data contains significant errors due to occasional failures and inaccuracies of the vision system. In particular, the physics simulator has difficulty modelling the frictional interactions of the real world and often is unable to accurately simulate a rotational movement of the object.

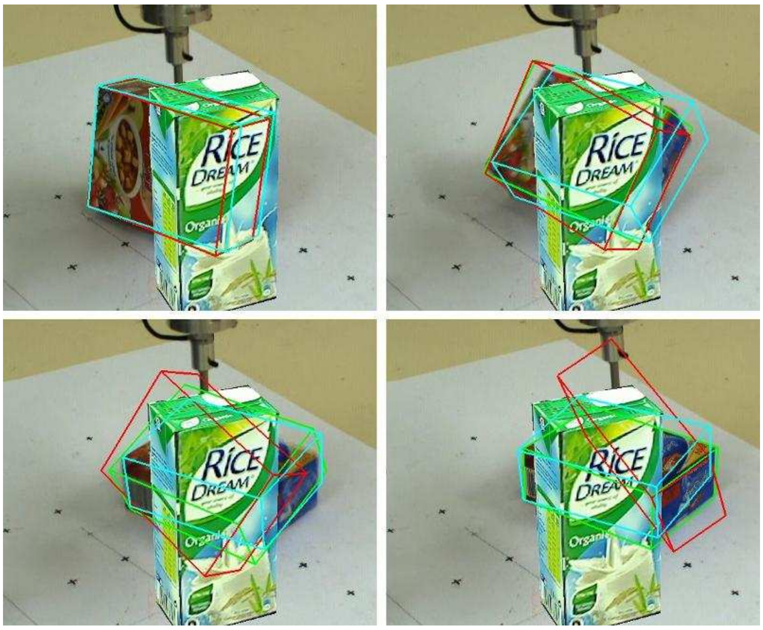


Figure 4.9: The toughest case: toppling combined with occlusion. (colour-based tracking with prediction: green, without prediction: red, pure prediction: blue)

Comparison of learning methods

We have trained the system on 9, 90 and 900 pushes of a polyflap object with a real robotic finger. We evaluated the performance of the multiple expert and regression methods. Figure 4.12 shows that the average and final prediction error decreases with the increased number of trials used in learning for both tested prediction methods. The multiple expert method performed reasonably well, even when trained on as little as 9 example pushes. The method performed particularly well with 90 learning trials, as local experts successfully prevented the predictor from violating impenetrability constraints that were frequently violated by the regression method. However, the performance of the multiple expert method did not significantly improve with 900 learning trials. We suspect a reason for that is the limited accuracy of the tracker (c.f. Table 3.1). For example, cases of tipping and toppling movements are particularly difficult to track, so that the prediction system does not always have sufficiently accurate training data to precisely learn all possible motions. To handle this problem we propose a deeper analysis.

Generalisation to unknown pushes

The proposed prediction technique is tested to which extent it can handle novel pushing configurations, which have not been learned previously. Therefore the predictors are trained with 900 trials of pushes applied to a polyflap in forwards direction. Then the predictors are tested on 100 pushes in backwards direction. Figure 4.13 and 4.14 show example frames from the prediction on simulated and real data. In both cases

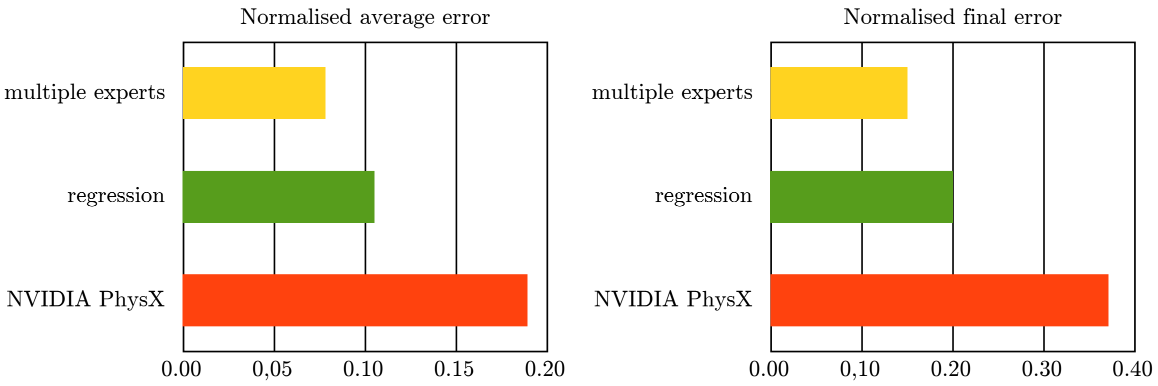


Figure 4.10: Physics simulation is unable to match the performance of learned predictors which have been trained in real experiments.

LWPR completely fails to generalise, as it predicts that the polyflap will not move at all, because it was never exposed to these kind of pushes. In contrast, the KDE product of experts is able to make physically plausible predictions. Of course they are far from perfect, but the direction is roughly correct when using global and local experts. Further learning trials will update the representation for the unknown arm-object-environment configuration and lead to more accurate results.

Generalising to unknown object shapes

The predictors are tested to which extent they can generalise to novel objects, which have different shapes from those learned during training. Therefore two experiments are designed:

1. *Training on a polyflap – testing on a box:* The predictors are trained on a polyflap object (900 pushes), and then tested on a box object (200 pushes). Figure 4.15 shows some examples of the results achieved. The box moves in the approximate direction as the polyflap, but again the predictions are not quite precise.

2. *Training on a box and a cylinder – testing on two rigidly connected cylinders:* Figure 4.16 shows the predictions when the experts are trained on a box (500 pushes) and a cylinder (100 pushes), which means they encode sliding, toppling and rolling. Since the contact information between the environment and the connected cylinders are neither considered by the global nor the local expert, the object penetrates through the ground plane. Using additional experts as in [48] fixes the problems which yields to a more plausible prediction.

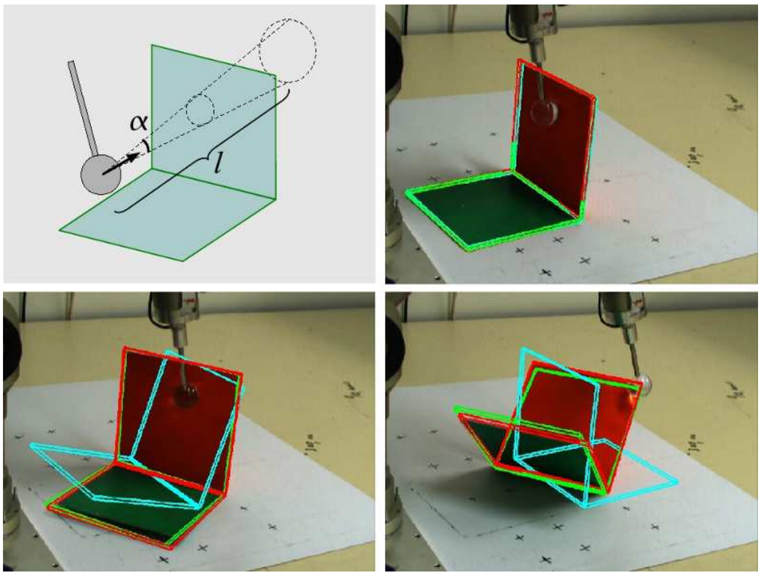


Figure 4.11: Random straight-line push with variable length $l = 25 \pm 5$ cm and angle $\alpha = 20$ degree towards an polyflap (top left). The object behaviour varies depending on the finger trajectory. In the image sequence the object begins to rotate before tilting. The red wire-frame shows the output of the tracking system. Green indicates the pose predicted by the proposed method, while the blue wire-frame is generated by the PhysX simulator.

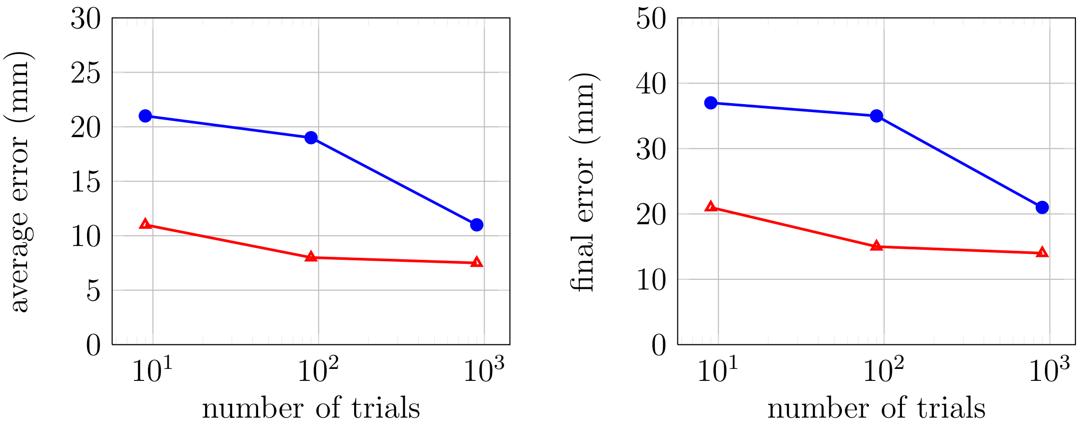


Figure 4.12: Decrease in average (left) and final (right) prediction errors with increasing number of learning trials. The prediction method *multiple experts* (red triangles) outperforms *regression* (blue dots).

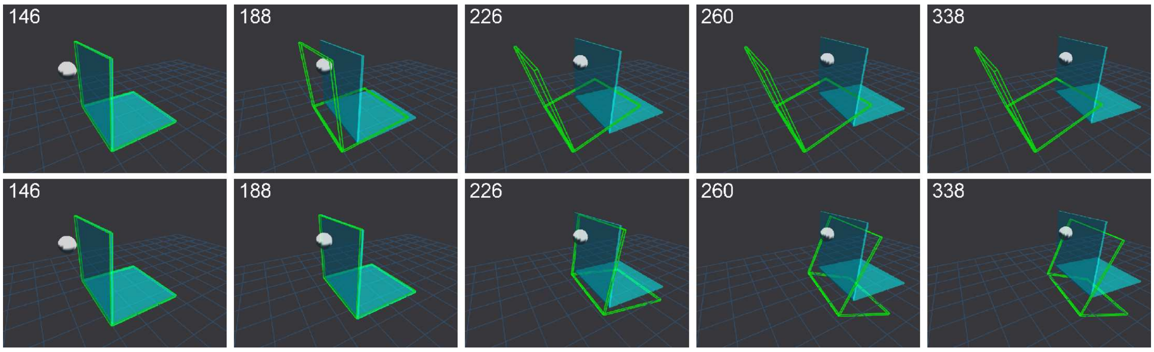


Figure 4.13: Generalisation to unknown pushes on simulated data. The green wire-frame shows the predictions (from top to bottom row) by KDE (global expert) and KDE (global and local expert) compared to simulated ground truth (in solid cyan). LWPR does not move at all and is therefore not shown.

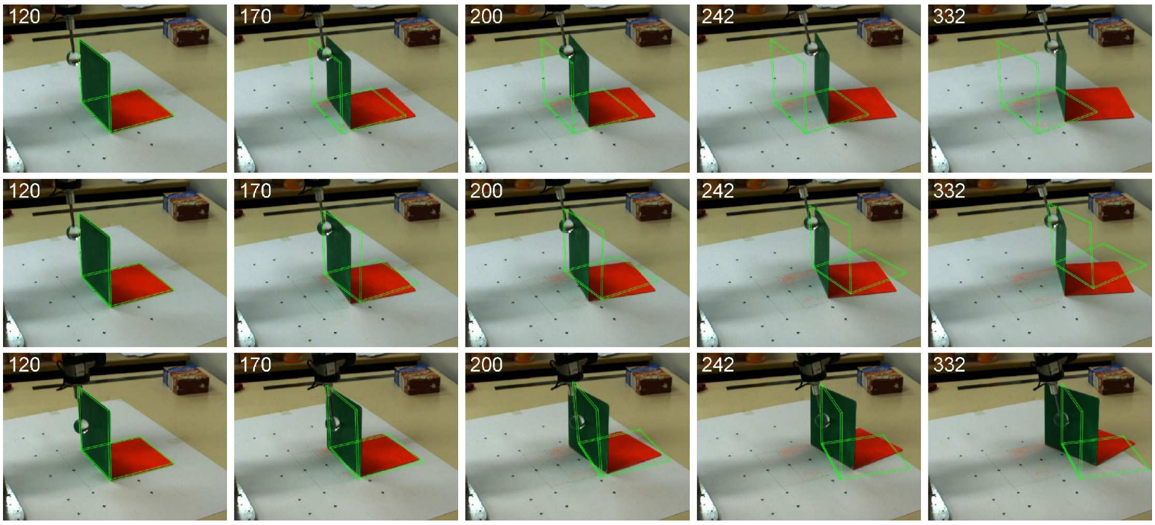


Figure 4.14: Generalisation to unknown pushes on real data. The green wire-frame shows the predictions (from top to bottom row) by KDE (global expert) and KDE (global and local expert) for one trial and by KDE (global and local expert) on another trial in which the polyflap rotates in a different direction as it slides. Note that the KDE (global expert) method predicts that the robot finger passes through the polyflap. LWPR does not move at all and is therefore not shown.

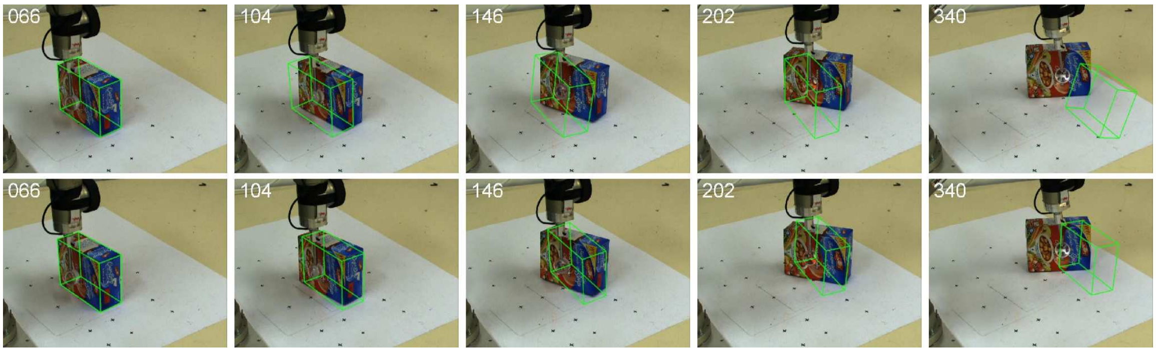


Figure 4.15: Generalisation to novel shape on real data. The green outline shows in the top row the predictions by KDE (global) and in the bottom row by KDE (global expert and local experts). LWPR does not move at all.

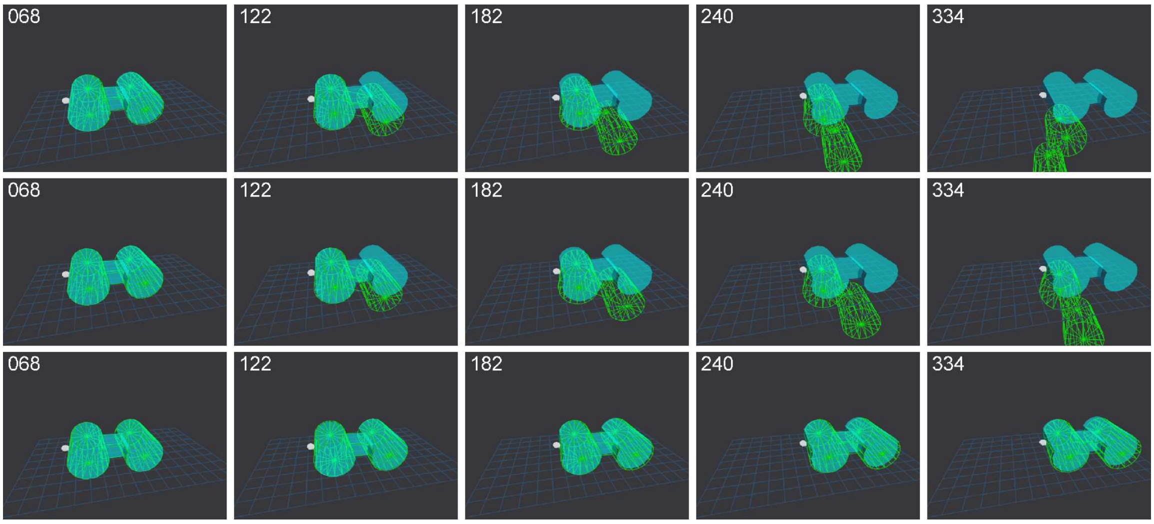


Figure 4.16: Generalisation to novel shape on simulated data. The green outline shows in the top row the predictions by KDE (global and local experts) and in the middle row by KDE (global expert). The bottom row shows the results when using more sophisticated experts as in [48] which prevents penetration through the ground plane. LWPR does not move at all.

4.6 Discussion

In this chapter we developed a probabilistic model to represent physical knowledge. Rigid body interactions are efficiently modelled by assigning probabilities in the transformation space of a certain pushing configuration. As future work we propose to apply the model completeness scheme of Section 3.3.4 based on this space. This would indicate if we have learned enough pushes and if not, which pushes to apply next.

Employing global and local experts increases the accuracy of predictions. Considering the shape learning methods outlined in Section 2.5 where only parts of the shape are modelled these experts may be introduced to the object the same way. However, the work on extension and generalisation of experts to new shapes is under development. Promising results have already been achieved and are presented in Section 4.5.

As demonstrated, the robustness of visual tracking of objects is significantly improved especially when it comes to difficult situations like heavy occlusion or fast movement causing motion blur. Further the accuracy of tracking increases since the prediction algorithm provides physical plausible poses. As a consequence of our whole system framework this improves the shape and colour models which in turn leads to better observations to update the predictors. Requirement for such a system to work is that all the models allow for extension and improvement.

Chapter 5

Conclusion

The methods and algorithms developed in this thesis aim to learn objects from scratch with respect to their visual appearance and physical behaviour. Therefore we follow a cycle of object segmentation, reconstruction, model tracking and physical interaction with the object. Any potential object segment can start the tracking process, with the idea to incrementally build up the object model. Once new information about the object arrives we immediately include it in our models, efficiently increasing the robustness and accuracy of all our algorithms.

5.1 Summary

Segmentation of images is by its nature an ill-posed problem. Additionally, colour and depth capturing devices typically introduce noise. Often range data is not aligned properly to the colour image, or worse has some areas where no data is available at all. Therefore we propose to use B-splines to overcome these problems. The usage of B-spline surfaces for segmentation implicitly poses the problem as the search for smooth, continuous surface patches. The implicit smoothing property of B-splines reduces noise both within continuous surface patches as well as at their boundaries. Our curve fitting method allows for alignment of range and image edges. When reconstructing objects from multiple views, where 3D data points from previous views can not be organized in a regular grid, our curve fitting algorithm can still efficiently compute the enclosing boundary of the surface. Further we can assign depth data, where none is given by the sensors, using the boundaries defined as B-spline curves on the B-spline surfaces. Another advantage is the simple and straight forward conversion of the trimmed B-splines to polygon meshes which are used for GPU acceleration of tracking.

We found that the key for any robotic learning approach to work is to observe changes of the object under inspection and reason about these observations. Although a large number of tracking algorithms have been proposed, barely any of them tackles the problem of a qualitative analysis of the current tracking state. To our best knowledge no solution, usable for learning object models, exists that solves this.

Therefore we have developed a visual tracking system that robustly reports the pose of the object together with qualitative statements of the observation. Using these statements we can now reliable identify views from which to learn new information and trigger or even guide the algorithms developed for learning.

Additionally, the tracking statements qualitatively classify the trajectories observed, which are used to train a physical prediction model. In return, these predictions are used to improve the performance of the tracking system, with respect to accuracy and robustness especially during difficult situations. Models for physical prediction of object movement are often very limited with respect to their generalisation to novel situations. Therefore we chose a probabilistic formulation developed by the University of Birmingham. In joint work we could show that we can now handle modifications in terms of object shape and in terms of scene configurations when an object is pushed by a robotic finger.

In conclusion, the thesis developed an approach to incrementally build up object models including shape, colour and physical behaviour. Tracking enables a robust and reliable flow of information. The repeated modelling steps allow to build up a more and more accurate and complete object model. Finally, we showed that the appearance model and tracking technique can be used to predict how objects move if poked.

5.2 Outlook

We could show that our segmentation method works for very difficult situations in strongly cluttered scenes. However, we do not account for occlusions and propose to extend our method with another SVM. It could be trained with respect to relations between object parts that indicate if they belong together or not. Considering our framework where objects are pushed by the robot we could use the resulting movement of the object to identify parts that belong together.

Our reconstruction approach models the shape of the object in order to visually track it and to model contact information during pushing manipulation. A lot of geometric modelling application and reverse engineering tools rely on watertight models. This property is not provided by our shape models. Using additional B-spline patches to model the seams between the surfaces could account for that. Another approach would be to use so called T-splines which allow to merge adjacent surface patches in a watertight manner.

Although pushing an object is a fundamental way to interact with objects, obviously at some point we want the robot to pick up objects which requires grasping. Therefore we propose to extend our method to multi-finger grasps and tests its promising characteristics. Once the object is grasped, the tutor-driven scenario, we assume throughout the thesis, could be omitted and replaced by self learning. Thereby the robot rotates the grasped object in front of its sensor, guided by our probabilistic formulation of model completeness, and learns the object from different views.

Finally, it would be interesting to see if the robot can in this way go out and learn all the objects in a certain environment. To do so, our visual tracking system has to be extended by an environment aware tracking system that allows to track objects even when they are out of view for a while. Errors introduced by the robot movement have to be taken into account leading to a SLAM like approach for reconstruction. Although a lot of work needs to be done, we think that we have laid the foundation towards such a scenario. Our message is clear: A successful being, be it robot or human, reasons carefully about all the knowledge arriving, tightly integrates it in its motivation and execution process and most importantly never stops learning.

Bibliography

[1] Y Aiyama, M Inaba, and H Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 136 –143 vol.1, 1993.

[2] Jonathan Balzer and Thomas Mörwald. Isogeometric Finite-Elements Methods and Variational Reconstruction Tasks in Vision A Perfect Match. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1624–1631, 2012.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. *European Conference on Computer Vision*, 3951(3):404–417, 2006.

[4] P J Besl and R C Jain. Segmentation Through Variable-Order Surface Fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):167–192, 1988.

[5] Georg Biegelbauer and Markus Vincze. Efficient 3D Object Detection by Fitting Superquadrics to Range Image Data for Robot's Object Manipulation. In *IEEE International Conference on Robotics and Automation*, pages 1086–1091, 2007.

[6] A Blake and M Isard. *Active Contours*, volume 17. Springer, 1998.

[7] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[8] D J Cappelleri, J Fink, B Mukundakrishnan, V Kumar, and J C Trinkle. Designing open-loop plans for planar micro-manipulation. In *IEEE International Conference on Robotics and Automation*, pages 637–642, 2006.

[9] J Chestnutt, S Kagami, K Nishiwaki, J Kuffner, and T Kanade. GPU-accelerated real-time 3D tracking for humanoid locomotion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[10] Tiberiu T. Cocias, Sorin M. Grigorescu, and Florin Moldoveanu. Multiple-superquadrics based object surface estimation for grasping in service robotics. In *Optimization of Electrical and Electronic Equipment*, pages 1471–1477, 2012.

[11] Alvaro Collet, Dmitry Berenson, Siddhartha S Srinivasa, and Dave Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Automation*, volume 27, pages 48–55, 2009.

[12] J A Cottrell, T J R Hughes, and Y Bazilevs. *Isogeometric Analysis - Toward Integration of CAD and FEA*. John Wiley & Sons, Ltd, 2009.

[13] M.G. Cox. The Numerical Evaluation of B-Splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.

[14] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In Holly Rushmeier, editor, *23rd Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH*, Annual Conference Series, pages 303–312, 1996.

[15] Timothy A Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.

[16] C. de Boor. On calculating with B-splines. *Journal of Approximation Theory*, 6:50–62, 1972.

[17] Hans de Ruiter and Beno Benhabib. Visual-model-based, real-time 3D pose tracking for autonomous navigation: methodology and experiments. *Autonomous Robots*, 25(3):267–286, 2008.

[18] H Delingette and J Montagnat. New Algorithms for Controlling Active Contours Shape and Topology. In *European Conference on Computer Vision*, pages 381–395. Springer, 2000.

[19] Arnaud Doucet, Nando De Freitas, and Neil Gordon, editors. *Sequential Monte Carlo methods in practice*. Springer, 2001.

[20] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[21] Stanimir Dragiev, Marc Toussaint, and Michael Gienger. Gaussian process implicit surfaces for shape estimation and grasping. In *IEEE International Conference on Robotics and Automation*, pages 2845–2850. Machine Learning group at the TU Berlin, Germany, 2011.

[22] T Drummond and R Cipolla. Real-Time Tracking of Complex Structures With on-Line Camera Calibration. In *British Machine Vision Conference*, pages 574–583, 1999.

[23] T Drummond and R Cipolla. Real-Time tracking of multiple articulated structures in multiple views. In *European conference on Computer Vision*, 2000.

[24] D Duff, J Wyatt, and R Stolkin. Motion estimation using physical simulation. In *IEEE International Conference on Robotics and Automation*, 2010.

[25] Damien Jade D.J. Duff, Thomas Mörwald, Rustam Stolkin, and Jeremy Wyatt. Physical simulation for monocular 3D model based tracking. In *IEEE International Conference on Robotics and Automation*, pages 5218–5225, May 2011.

[26] R Eskenazi and R T Cunningham. Real-Time Tracking of Moving Objects in TV Images. *IEEE Workshop Pattern Recognition and Artificial Intelligence*, pages 4–6, April 1978.

[27] Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cortography. *Communications of the ACM*, 24(6):381–395, 1981.

[28] S Flory. Fitting curves and surfaces to point clouds in the presence of obstacles. *Computer Aided Geometric Design*, 26(2):192–202, 2009.

[29] J Fuentes-Pacheco, J Ruiz-Ascencio, and J M Rendón-Mancha. Binocular visual tracking and grasping of a moving object with a 3D trajectory. *Journal of Applied Research and Technology*, 7(03):259–274, 2009.

[30] Yasutaka Furukawa and Jean Ponce. Carved Visual Hulls for Image-Based Modeling. *International Journal of Computer Vision*, 81(1):53–67, 2008.

[31] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–76, 2010.

[32] D Gennery. Visual tracking of known three-dimensional object. *International Journal of Computer Vision*, 1992.

[33] Iryna Gordon and David G Lowe. What and Where: 3D Object Recognition with Accurate Pose. *Toward Category-Level Object Recognition, J. Ponce, M. Hebert, Schmid. C., and A. Zisserman*, pages 67–82, 2006.

[34] N J Gordon, D J Salmond, and A F M Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEEE Radar and Signal Processing IEE Proceedings F*, 140(2):107–113, 1993.

[35] Feng Han Feng Han, Zhuowen Tu Zhuowen Tu, and Song-Chun Zhu Song-Chun Zhu. Range image segmentation by an effective jump-diffusion method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1138–1153, 2004.

[36] C Harris and A Blake ed. Tracking with Rigid Bodies. *Active Vision*, pages 59–73, 1992.

[37] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-Time Plane Segmentation using RGB-D Cameras. In *RoboCup Symbosium*, number D, pages 230–239. Eurographics Association, 2011.

[38] Dirk Holz and Behnke Sven. Fast Range Image Segmentation and Smoothing using Approximate Surface Reconstruction and Region Growing. In *12th International Conference on Intelligent Autonomous Systems*, 2012.

[39] Ales Jaklic, Ales Leonardis, and Franc Solina. *Segmentation and Recovery of Superquadrics.* Kluwer, 2000.

[40] D Katsoulas, C C Bastidas, and D Kosmopoulos. Superquadric Segmentation in Range Images via Fusion of Region and Boundary Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):781–95, 2008.

[41] Stefan Klanke, Sethu Vijayakumar, and Stefan Schaal. A Library for Locally Weighted Projection Regression. *Journal of Machine Learning Research*, 9:623–626, 2007.

[42] G Klein and D Murray. Full-3D Edge Tracking with a Particle Filter. In *British Machine Vision Conference*, 2006.

[43] Georg Klein and Tom Drummond. Robust Visual Tracking for Non-Instrumented Augmented Reality. In *IEEE/ACM International Symposium on Mixed and Augmented Reality*, 2003.

[44] D Koller, K Daniilidis, and H.-H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 1993.

[45] M Kopicki. *Prediction learning in robotic manipulation.* PhD thesis, University of Birmingham, 2010.

[46] M Kopicki, J Wyatt, and R Stolkin. Prediction learning in robotic pushing manipulation. In *International Conference on Advanced Robotics*, pages 1–6, 2009.

[47] Marek Kopicki, Rustam Stolkin, Sebastian Zurek, Thomas Mörwald, and Jeremy Wyatt. Predicting workpiece motions under pushing manipulations using the principle of minimum energy. In *Robotics: Science and Systems, workshop*, 2009.

[48] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Mörwald, and Jeremy Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *IEEE International Conference on Robotics and Automation*, pages 5722–5729, May 2011.

[49] A Kosaka and G Nakazawa. Vision-based motion tracking of rigid objects using prediction of uncertainties. *IEEE International Conference on Robotics and Automation*, 1995.

[50] Danica Kragic, Andrew T Miller, and Peter K Allen. Real-time tracking meets online grasp planning. In *IEEE International Conference on Robotics and Automation*, pages 2460–2465, 2001.

[51] Geert-Jan M. Kruijff, John D. Kelleher, Gregor Berginc, and Ale\v{s} Leonardis. Structural descriptions in human-assisted robot visual learning. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 343–344, 2006.

[52] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust Object Detection with Interleaved Categorization and Segmentation. *International Journal of Computer Vision*, 77(1-3):259–289, 2007.

[53] A Leonardis, A Jaklic, and F Solina. Superquadrics for segmenting and modeling range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1289–1295, 1997.

[54] Aleš Leonardis, Alok Gupta, and Ruzena Bajcsy. Segmentation of range images as the search for geometric parametric models. *International Journal of Computer Vision*, 14(3):253–277, April 1995.

[55] Hector Levesque and Gerhard Lakemeyer. Cognitive Robotics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 23, pages 869–886. Elsevier, 3 edition, 2008.

[56] Hector Levesque and R. Reiter. Beyond planning. In *AAAI Spring Symbosium on Integrating Robotics Research*, Palo Alto, 1998.

[57] Jianguo Li Jianguo Li, E Li, Yurong Chen Yurong Chen, Lin Xu Lin Xu, and Yimin Zhang Yimin Zhang. Bundled depth-map merging for multi-view stereo. In *IEEE Computer Vision and Pattern Recognition*, volume 24, pages 2769–2776, 2010.

[58] Yi Li, Jean-Philippe Saut, Juan Cortes, Thierry Simeon, and Daniel Sidobre. Finding enveloping grasps by matching continuous surfaces. In *IEEE International Conference on Robotics and Automation*, pages 2825–2830, 2011.

[59] M I A Lourakis and A A Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30, 2009.

[60] D G Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, pages 113–122, 1992.

[61] David G Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[62] K Lynch. The mechanics of fine manipulation by pushing. In *IEEE International Conference on Robotics and Automation*, pages 2269–2276, 1992.

[63] Kevin M Lynch. Toppling Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 152–159, 1999.

[64] E Marchand and P Bouthemy. A 2D-3D Model-based Approach to Real-time Visual Tracking. *Image and Vision Computing*, 19:941–955, 2001.

[65] M T Mason. Manipulator grasping and pushing operations. In *PhD thesis MIT*. Massachusetts Institute of Technology (MIT), 1982.

[66] Lucie Masson, Michel Dhome, and Frederic Jurie. Robust Real Time Tracking of 3D Objects. In *International Conference on Pattern Recognition*, 2004.

[67] Robert Mc Neel and Associates. The openNURBS Initiative, a C++ source code library - www.opennurbs.org, 2011.

[68] Philipp Michel, Joel Chestnutt, Satoshi Kagami, Koichi Nishiwaki, James Kuffner, and Takeo Kanade. GPU-accelerated Real-Time 3D Tracking for Humanoid Autonomy. In *JSME Robotics and Mechatronics Conference*, June 2008.

[69] A T Miller, S Knoop, H I Christensen, and P K Allen. Automatic grasp planning using shape primitives. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1824–1829. IEEE, 2003.

[70] Pradit Mittrapiyanuruk, Guilherme N Desouza, and Avinash C Kak. Accurate 3D tracking of rigid objects with occlusion using active appearance models. In *7th IEEE Workshop on Applications of Computer Vision / IEEE Workshop on Motion and Video Computing*, pages 90–95, 2005.

[71] Thomas Mörwald, Marek Kopicki, Rustam Stolkin, Jeremy Wyatt, Sebastian Zurek, Michael Zillich, and Markus Vincze. Predicting the Unobservable, Visual 3D Tracking with a Probabilistic Motion Model. In *IEEE International Conference on Robotics and Automation*, pages 1849–1855, May 2011.

[72] Thomas Mörwald, Johann Prankl, Andreas Richtsfeld, Michael Zillich, and Markus Vincze. BLORT - The Blocks World Robotic Vision Toolbox. In *IEEE International Conference on Robotics and Automation, Workshop*, 2010.

[73] Thomas Mörwald, Andreas Richtsfeld, Johann Prankl, Michael Zillich, and Markus Vincze. Geometric data abstraction using B-splines for range image segmentation. In *IEEE International Conference on Robotics and Automation*, Karlsruhe, 2013.

[74] Thomas Mörwald, Michael Zillich, Johann Prankl, and Markus Vincze. Self-Monitoring to Improve Robustness of 3D Object Tracking for Robotics. In *IEEE International Conference on Robotics and Biomimetics*, 2011.

[75] Thomas Mörwald, Michael Zillich, and Markus Vincze. Edge tracking of textured objects with a recursive particle filter. In *Proceedings of the GraphiCon*, Moscow, Russia, 2009.

[76] E Murphy-Chutorian and Mohan M Trivedi. Particle Filtering with Rendered Models: A Two Pass Approach to Multi-Ojbect 3D Tracking with the GPU. In *Computer Vision and Pattern Recognition, Workshop*, 2008.

[77] H H. Nagel. Representation of Moving Rigid Objects Based on Visual Observations. *Computer*, 14(8):29–39, August 1981.

[78] Mustafa Özuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast Keypoint Recognition using Random Ferns. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[79] Qi Pan, Gerhard Reitmayr, and Tom Drummond. ProFORMA : Probabilistic Feature-based On-line Rapid Model Acquisition. *20th British Machine Vision Conference*, (c):1–11, 2009.

[80] H. Park. Choosing nodes and knots in closed B-spline curve interpolation to point data. *Computer-Aided Design*, 35(1):123, 2000.

[81] M A Peshkin and A C Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation*, 4:569–598, 1988.

[82] Les Piegl and Wayne Tiller. *The NURBS book*. Monographs in visual communication. Springer, 1996.

[83] Andreas Richtsfeld, Thomas Mörwald, Johann Prankl, Michael Zillich, and Markus Vincze. Segmentation of Unknown Objects in Indoor Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[84] Andreas Richtsfeld, Thomas Mörwald, Michael Zillich, and Markus Vincze. Taking in shape: Detection and tracking of basic 3d shapes in a robotics context. In *Computer Vision Winter Workshop*, pages 91–98, 2010.

[85] Andreas Richtsfeld, Johann Prankl, Jonathan Balzer, and Michael Zillich. Towards Scene Understanding Object Segmentation Using RGBD-Images. In *Computer Vision Winter Workshop*, 2012.

[86] A Ruf, M Tonko, R Horaud, and H.-H. Nagel. Visual tracking by adaptive kinematic prediction. In *International Conference on Intelligent Robots and Systems*, 1997.

[87] R B Rusu, A Holzbach, R Diankov, G Bradski, and M Beetz. Perception for mobile manipulation and grasping using active stereo. In *9th IEEE/RAS International Conference on Humanoid Robots*, number May, pages 632–638, 2009.

[88] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.

[89] J R Sánchez, H Álvarez, and D Borro. Towards real time 3D tracking and reconstruction on a GPU using Monte Carlo simulations. In *IEEE International Symposium on Mixed and Augmented Reality*, pages 185–192, 2010.

[90] Sudeep Sarkar and Kim L Boyer. Perceptual organization in computer vision - A review and a proposal for a classificatory structure. *IEEE Transactions On Systems Man And Cybernetics*, 23(2):382–399, 1993.

[91] D. Skocaj, G. Berginc, B. Ridge, A. Stimec, M. Jogan, O. Vanek, A. Leonardis, M. Hutter, and N. Hawes. A system for continuous learning of visual concepts. In *International Conference on Computer Vision Systems*, 2007.

[92] Michael Stark, Philipp Lies, Michael Zillich, Jeremy Wyatt, and Bernt Schiele. Functional Object Class Detection Based on Learned Affordance Cues. In Antonios Gasteratos, Markus Vincze, and JohnK Tsotsos, editors, *Computer Vision Systems*, pages 435–444. Springer Berlin Heidelberg, 2008.

[93] R Storn and K Price. Differential evolution. A simple and efficient heuristic for global optimization over continuous spaces. In *Journal of Global Optimization*, volume 11, pages 341–359, 1997.

[94] L Vacchetti, V Lepetit, and P Fua. Stable Real-Time 3D Tracking using Online and Offline Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[95] Luca Vacchetti, Vincent Lepetit, and Pascal Fua. Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking. In *IEEE/ACM International Symposium on Mixed and Augmented Reality*, 2004.

[96] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005.

[97] M Vincze, M Ayromlou, W Ponweiser, and M Zillich. Edge-Projected Integration of Image and Model Cues for Robust Model-Based Object Tracking. *The International Journal of Robotics Research*, 2001.

[98] George Vogiatzis, Carlos Hernández Esteban, Philip H S Torr, and Roberto Cipolla. Multiview stereo via volumetric Graph-Cuts and occlusion robust photo-consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2241–2246, 2007.

[99] Wenping Wang, Helmut Pottmann, and Yang Liu. Fitting B-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics*, 25(2):214–238, 2006.

[100] M Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychological Research*, 4(1):301–350, 1923.

[101] Jianhua Wu and Leif Kobbelt. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[102] P Wunsch and G Hirzinger. Registration of CAD-Models to Images by Iterative Inverse Perspective Matching. In *13th International Conference on Pattern Recognition*, pages 77–83, 1996.

[103] Huaiping Yang, Wenping Wang, and Jiaguang Sun. Control point adjustment for B-spline curve approximation. *Computer-Aided Design*, 36(7):639–652, 2004.

[104] Christopher Zach. Fast and High Quality Fusion of Depth Maps. *4th International Symposium on 3D Data Processing, Visualization and Transmission*, 1:1–8, 2008.

[105] Michael Zillich, Johann Prankl, Thomas Mörwald, and Markus Vincze. Knowing your limits - self-evaluation and prediction in object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 813–820. Automation and Control Institute, Vienna University of Technology, Austria, 2011.

Curriculum Vitae

Personal data

Full Name:	Thomas Mörwald
Academic Degree:	Dipl.-Ing.
Address:	Reithlegasse 15/3, 1190 Wien, Austria
Date and place of birth:	16^{th} May 1982, Steyr, Austria
Citizenship:	Austrian
Homepage:	`http://users.acin.tuwien.ac.at/tmoerwald`

Education

2002 - 2008:	Diploma Study of Mechatronics Specialisation in Computer Science and Robotics Johannes Kepler University, Linz.
1996 - 2001:	Secondary Technical College for Mechanical Engineering (HTL), Steyr.

Work Experience

2008 - 2009:	Research and development for pneumatic robots, FerRobotics Compliant Robot Technology GmbH, Linz

Research projects

Feb. 2009 - Jun. 2012:	*Cognitive Systems that Self-Understand and Self-Extend* (CogX)
Okt. 2012 - Okt. 2014:	*Interactive Modelling of Daily-life Objects* (inMODO)

Publications

T. Mörwald, A. Richtsfeld, J. Prankl, M. Zillich, M. Vincze: *"Geometric data abstraction using B-splines for range image segmentation"; Int. Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013.*

A. Richtsfeld, T. Mörwald, J. Prankl, M. Zillich, M. Vincze: *"Segmentation of Unknown Objects in Indoor Environments"; Int. Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, 2012.*

J. Balzer, T. Mörwald: *"Isogeometric Finite-Elements Methods and Variational Reconstruction Tasks in Vision - A Perfect Match"; Conference on Computer Vision and Pattern Recognition (CVPR), Providence, Rhode Island, USA, 2012.*

A. Richtsfeld, T. Mörwald, J. Prankl, J. Balzer, M. Zillich, M. Vincze: *"Towards Scene Understanding - Object Segmentation Using RGBD-Images"; Computer Vision Winter Workshop (CVWW), Mala Nedelja, Slovenia, 2012.*

M. Zillich, J. Prankl, T. Mörwald, M. Vincze: *"Knowing Your Limits - Self-Evaluation and Prediction in Object Recognition"; Int. Conference on Intelligent Robots and Systems (IROS), San Francisco, USA, 2011.*

T. Mörwald, M. Zillich, J. Prankl, M. Vincze: *"Self-Monitoring to Improve Robustness of 3D Object Tracking for Robotics"; Int. Conference on Robotics and Biomimetics (ROBIO), Thailand, 2011.*

D. Duff, T. Mörwald, R. Stolkin, J. Wyatt: *"Physical simulation for monocular 3D model based tracking"; Int. Conference on Robotics and Automation (ICRA), Shanghai, 2011.*

M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, J. Wyatt: *"Learning to predict how rigid objects behave under simple manipulation"; Int. Conference on Robotics and Automation (ICRA), Shanghai, 2011.*

T. Mörwald, M. Kopicki, R. Stolkin, J. Wyatt, S. Zurek, M. Zillich, M. Vincze: *"Visual 3D Tracking with a Probabilistic Motion Model"; Int. Conference on Robotics and Automation (ICRA), Shanghai, 2011.*

K. Sjöö, A. Aydemir, T. Mörwald, K. Zhou, P. Jensfelt: *"Mechanical support as a spatial abstraction for mobile robots"; Int. Conference on Intelligent Robots and Systems (IROS), Taipai, Taiwan, 2010.*

T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, M. Vincze: *"BLORT- The Blocks World Robotic Vision Toolbox"; IEEE International Conference on Robotics and Automation (workshop, ICRA), Anchorage, Alaska, 2010.*

A. Richtsfeld, T. Mörwald, M. Zillich, M. Vincze: *"Taking in Shape: Detection and Tracking of Basic 3D Shapes in a Robotics Context"; Computer Vision Winter Workshop (CVWW), Nove Hrady, Czech Republic, 2010.*

M. Kopicki, R. Stolkin, S. Zurek, T. Mörwald, J. Wyatt: *"Predicting workpiece motions under pushing manipulations using the principle of minimum energy"; Robotics: Science and Systems Workshop, Zaragoza, Spanien, 2010.*

T. Mörwald, M. Zillich, M. Vincze: *"Edge Tracking of Textured Objects with a Recursive Particle Filter"; Int. Conference on Computer Graphics and Vision (GraphiCon), Moskau, Russland, 2009.*

T. Mörwald, K. Stadlbauer, H. Bremer: *"Simulation and 3D-visualisation of mechanical systems with Linux RTAI"; Diploma Thesis, Johannes Kepler University Linz, Institute for Robotics, Austria, 2008*