Techniques for Robot Motion Planning in Environments with Deformable Objects

Barbara Frank

Technische Fakultät
Albert-Ludwigs-Universität Freiburg im Breisgau

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Wolfram Burgard

April 2013

Techniques for Robot Motion Planning in Environments with Deformable Objects

Barbara Frank

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan:	Prof. Dr. Yiannos Manoli
Erstgutachter:	Prof. Dr. Wolfram Burgard
Zweitgutachter:	Prof. Dr. Matthias Teschner
Tag der Disputation:	08.04.2013

Zusammenfassung

Mobile Roboter erfreuen sich in den letzten Jahren immer größerer Beliebtheit und werden heute schon für eine Vielzahl von Aufgaben eingesetzt. Als Beispiele seien Service-Roboter oder das Google-Auto genannt. Diese Systeme müssen in der Lage sein, ihre Bewegungen zu planen, um andere sinnvolle Aufgaben zu erfüllen. Dies wiederum setzt ein Modell der Umgebung voraus. Die Umgebung wird üblicherweise als statisch beziehungsweise als nicht verformbar angenommen. In der echten Welt kommen jedoch auch verschiedene verformbare Hindernisse vor, beispielsweise Vorhänge oder Pflanzen, die einem Roboter den Weg versperren können. Ist ein Roboter in der Lage, solche Verformungseigenschaften bei der Pfadplanung zu berücksichtigen, kann er ein Ziel möglicherweise viel effizienter erreichen, zum Beispiel, indem er den Vorhang beiseite schiebt anstatt einen Umweg zu nehmen. Die Berücksichtigung solcher Verformungseigenschaften erhöht jedoch die Komplexität sowohl der Umgebungsmodellierung als auch des Pfadplanungsproblems.

Ziel dieser Arbeit ist es, zu erforschen, wie Roboter sich in Umgebungen mit verformbaren Hindernissen zurechtfinden können. In diesem Zusammenhang ergeben sich verschiedene interessante Fragestellungen:

(1) Wie kann ein Roboter etwas über die Verformungseigenschaften von Objekten lernen?

(2) Wie kann ein Roboter Verformungen von Hindernissen bei der Bewegungsplanung möglichst effizient berücksichtigen?

(3) Wie kann ein Roboter geplante Bewegungen sicher und zuverlässig ausführen?

Wir stellen Techniken und Ansätze vor, welche diese Fragestellungen mit echten Robotern lösen können.

Um Verformungsmodelle von Objekten zu bestimmen, wurde ein Manipulationsroboter mit einer Tiefenkamera und einem Kraft-Momenten-Sensor ausgestattet. Der Roboter kann ein Objekt aus verschiedenen Blickwinkeln beobachten und die erhaltenen Scans in ein dreidimensionales Modell des Objekts registrieren. Um die Elastizitätsparameter eines Objekts zu bestimmen, interagiert der Roboter mit dem Objekt, und misst gleichzeitig die auf das Objekt ausgeübte Kraft sowie die daraus resultierende Verformung. Diese Beobachtungen werden mit der Simulation eines Finite-Element-Modells verglichen. Die Parameter des Modells werden dabei mittels eines Gradienten-Verfahrens optimiert, so dass der Fehler zwischen simulierter Verformung und tatsächlich beobachteter Verformung minimiert wird.

Die gelernten Verformungsmodelle können in einer physikalischen Simulation eingesetzt werden, um die Auswirkungen auf Hindernisse beziehungsweise die Kosten verschiedener Roboterbewegungen in der Pfadplanung zu berücksichtigen. Solche Simulationen sind jedoch sehr zeitaufwendig und für die Planung eines Pfades werden üblicherweise viele Alternativen in Betracht gezogen. Ein Pfadplanungsystem, das auf echten Robotern eingesetzt wird, sollte in der Lage sein, Pfadanfragen möglichst effizient zu beantworten, um eine schnelle Reaktion des Roboters auf neue Aufgaben zu ermöglichen. Daher kommt die Durchführung von Simulationen zur Anfragezeit nicht in Frage. Um die Bewegungsplanung effizienter zu gestalten, führen wir Kostenfunktionen für verformbare Objekte ein. Wir nehmen an, dass Objekte zwar verformt aber nicht bewegt werden können, wie beispielsweise ein Vorhang, der an der Vorhangstange befestigt ist. Außerdem berücksichtigen wir nur Interaktionen zwischen dem Roboter und einzelnen Hindernissen, aber keine Interaktionen zwischen verschiedenen Hindernissen. Dies ermöglicht uns die Vorberechnung einer Kostenfunktion für jedes Objekt, welche die Verformungskosten von Roboter-Trajektorien relativ zum Objekt modelliert. Als Maß für die Verformungskosten betrachten wir dabei die potentielle Energie des Objektes und wir verwenden Gauß-Prozesse, um Kostenfunktionen zu modellieren. Dieser nicht-parametrische Ansatz zur Regression modelliert eine Wahrscheinlichkeitsverteilung über Funktionen auf Basis einer Menge von Trainingsdaten. Die Trainingsdaten sind in unserem Fall Roboter-Trajektorien und die zugehörigen Verformungskosten. Diese können in einem Vorberechnungsschritt durch entsprechende Simulationen erzeugt werden. Bei der Suche nach einem Pfad zu einem Zielpunkt können die Verformungskosten beliebiger Trajektorien durch Regression bestimmt werden.

Wir stellen einen Planungsansatz vor, der auf *Probabilistic Roadmaps* basiert. In diesem *sampling-basierten* Ansatz werden erlaubte Roboterkonfigurationen in einer Umgebung und mögliche Pfade zwischen diesen Konfigurationen durch einen Graphen repräsentiert. Wir verwenden die vorgestellten Kostenfunktionen, um die Verformungskosten von Pfadsegmenten in der Roadmap zu evaluieren. Ein Weg zu einem Zielpunkt kann dann durch eine Graphensuche bestimmt werden. In unserer Anwendung wird der Pfad mit dem besten Verhältnis zwischen Verformungskosten und Wegekosten bestimmt.

Nachdem ein Weg zu einem vorgegebenen Zielpunkt bestimmt wurde, muss dieser vom Roboter ausgeführt werden. Dabei ergeben sich neue Herausforderungen. Zum einen muss der Roboter mit Unsicherheiten in seiner Positionsschätzung oder mit möglichen Abweichungen von der geplanten Trajektorie umgehen können. Zum anderen kann der Roboter dynamischen Hindernissen begegnen, die nicht in seinem Modell enthalten sind. Dieses

Problem spielt eine wichtige Rolle in Umgebungen, die auch von Menschen bevölkert sind. Sensor-basierte Ansätze zur Kollisionsvermeidung sorgen in nicht-verformbaren Umgebungen für eine sichere Pfadausführung. Auf verformbare Umgebungen lässt sich ein solcher Ansatz jedoch nicht direkt übertragen, da Kontakt mit verformbaren Hindernissen durchaus erlaubt ist. Der Roboter muss also in der Lage sein, erlaubte Kollisionen mit Hindernissen, die verformt werden müssen, von bevorstehenden Kollisionen mit nicht-verformbaren Hindernissen zu unterscheiden. Wir stellen ein probabilistisches Sensormodell vor, welches, gegeben die Position und aktuelle Beobachtung des Roboters, die Wahrscheinlichkeit bestimmt, dass eine Messung zu einem verformbaren Hindernis gehört. Dieses Modell ermöglicht dem Roboter, seine Sensor-Messungen entsprechend zu interpretieren und Pfade sicher und zuverlässig auszuführen.

Das von uns entwickelte Planungssystem wurde auf zwei unterschiedlichen Robotern eingesetzt – auf einem Manipulationsroboter mit sieben Freiheitsgraden und auf einem Radroboter. Wir zeigen in verschiedenen Anwendungsbeispielen, dass Roboter mit unserem Ansatz erfolgreich in Umgebungen mit verformbaren Hindernissen navigieren können und auch Ziele erreichen können, für die bisherige Planer keine Lösung finden.

Abstract

The ability to plan motions is essential for autonomous robots that accomplish high-level tasks. Planning, in turn, requires an appropriate model of the environment. Real-world environments contain various deformable obstacles, for instance curtains or plants, and a robot can possibly deform these objects in order to reach a goal. This is relevant, for instance, for household and service robots, or for robotic surgical assistants. Most planning approaches, however, assume the environment to be rigid, since taking into account object deformations considerably increases the complexity of the modeling as well as the planning problem.

In this thesis, we investigate how robots can find their way around in environments containing deformable obstacles. We study several subproblems that arise in this context, namely (1) the acquisition of deformation models, (2) efficient planning techniques considering these models, and (3) a safe and reliable execution of planned motions.

We first present an approach that allows a robot to model deformable objects. The robot interacts with objects and collects data of the resulting deformations. Material parameters are estimated by optimizing a finite element model with respect to the observations. The learned deformation models can then be used in physical simulations to evaluate the effects of different robot motions. We present a planning framework that optimizes the trade-off between motion costs and the costs introduced by deforming objects. Carrying out the corresponding simulations during planning time, however, is time-consuming. Therefore, we present an approach to model object deformation cost functions based on Gaussian process regression, which can be efficiently evaluated when answering path queries. For stationary objects, the simulations of robot motions to generate data for regression can be done in a preprocessing step. Finally, when executing a planned motion, robots need to be prepared to avoid collisions with unforeseen obstacles based on their sensor measurements. This is of particular importance in environments populated by humans. When deforming an object, however, contact is inevitable. We therefore introduce a probabilistic sensor model for deformable objects that enables the robot to distinguish allowed contacts from collisions that need to be avoided. In this way, we ensure a safe and successful path execution.

We implemented our approaches on real robots and applied the developed planning framework to different platforms, a wheeled robot and a manipulator with many degrees of freedom. In several experiments, we demonstrated that our robots are able to successfully navigate in environments with deformable objects and that they can accomplish tasks going beyond the capabilities of traditional planners designed for rigid environments.

Acknowledgments

Writing a PhD thesis is an enormous project and would not have been possible without the help and encouragement of many people, whom I would like to thank with all my heart.

First of all, I would like to thank my supervisor Wolfram Burgard for giving me the great opportunity to pursue my PhD in his lab, for the chance to work on an exciting project and for providing an excellent working environment. I much appreciate how he takes care of all his students and how he always makes time to discuss progress and open problems and to provide valuable ideas, suggestions and encouragement.

I am very thankful to Cyrill Stachniss for being such a committed co-supervisor and a constant source of inspiration, for always taking time to discuss problems and possible solutions, for providing feedback, and for finishing papers late at night, all in addition to being a great friend. I would like to thank him for the many things I learned under his guidance.

I would like to thank Matthias Teschner for the great collaboration within our joint research project, for many enlightening discussions, particularly in the beginning of my work in the field, and for agreeing to be my co-examiner. I would also like to thank Maren Bennewitz and Bernhard Nebel for taking time to be on my thesis committee.

I would like to thank my project partners from the computer graphics group for providing me with their great simulation engine and letting me in on its secrets. In particular, I would like to thank Markus Becker and Rüdiger Schmedding for many hours of discussions and experiments and our fruitful collaboration that resulted in several joint publications. Many thanks also to Sebastian Schulz for porting the simulation framework to Linux.

Many thanks go to Jürgen Sturm for his ideas, enthusiasm and technical expertise regarding the design and construction of our mobile manipulation robot Zora. Without him, this robot would probably not be moving a limb. I would also like to say thanks to the great Hiwis Lionel Ott and Nichola Abdo for keeping the robot up and running, Jörg Müller for always coming up with the right tools to figure out hardware problems and for going out of his way to fix them, furthermore Jonas Rist and Benedikt Frank for their help with various hardware issues. A special thanks goes to Axel Rottmann for sharing his expertise and his developed software framework on Gaussian process regression.

I would like to say thanks to Patrick Pfaff, Maren Bennewitz, Jeff Trinkle, and Marija Dakulović for being great office mates, for sharing fun and work-intensive times and the occasional chocolate.

I would like to thank all my colleagues and friends in the AIS lab for the great and open-minded atmosphere, for sharing ideas and nice pieces of software, for countless discussions about research and all the rest, and for making my PhD time such an enjoyable experience. I really enjoyed our tea and fruit breaks, barbecues, table tennis matches, pizza-sessions before paper deadlines, and our trips to conferences and workshops in interesting places all over the world.

Many thanks to Susanne Bourjaillat, Dagmar Sonntag, Michael Keser, and Peter Winterer for their support with all sorts of administrative and technical issues. They made life a lot easier.

Many thanks go to my "scientific reviewers", Daniel Mader, Christoph Sprunk, Thomas Hippler, Maximilian Beinhofer, Michael Ruhnke, Henrik Kretzschmar, Cyrill Stachniss, Markus Kuderer, Maren Bennewitz, Katharina Frank, Marei Hopert, Jeff Trinkle, Felix Endres, Nichola Abdo, and Jed Williams for carefully proof-reading earlier versions of this document and providing valuable comments and suggestions. All remaining errors and obscurities are entirely up to me.

A special thanks goes to my friends for always sharing my joys and sorrows and for keeping me down to earth. Finally, I would like to express my deepest gratitude to my family for their unconditional love and support.

This work has been supported by the German Research Foundation (DFG) under contract number SFB-TR8. Their support is gratefully acknowledged.

Contents

1 Introduction **1**
1.1 Contributions of this Thesis . . . 5
1.2 Publications . . . 6
1.3 Collaborations . . . 7

2 Overview **9**

3 Background **11**
3.1 Deformation Simulation . . . 11
3.1.1 Collision Detection . . . 12
3.1.2 Computation of Contact Forces . . . 13
3.1.3 Time Integration . . . 17
3.2 Deformation Model . . . 18
3.2.1 Elasticity Parameters . . . 19
3.2.2 Linear Finite Element Approximation . . . 21
3.3 Summary . . . 25

4 Learning Deformation Models **27**
4.1 Data Acquisition . . . 28
4.1.1 The Robotic System . . . 28
4.1.2 Geometric Models for Simulation . . . 29
4.1.3 Deformation of Objects . . . 34
4.2 Parameter Estimation . . . 35
4.2.1 FEM Simulation . . . 37
4.2.2 Error Function . . . 37
4.2.3 Parameter Optimization . . . 38
4.3 Limitations of the Deformation Model . . . 40
4.4 Experimental Results . . . 42
4.4.1 Simulation Experiments . . . 42
4.4.2 Parameter Estimation for Real Objects . . . 44
4.5 Summary . . . 53

5 Deformation Cost Functions for Motion Planning **55**
5.1 Deformation Costs of a Robot Trajectory . . . 55
5.2 Object Deformation Cost Functions . . . 56

5.3 Modeling Deformation Cost Functions with Gaussian Processes 59
5.3.1 Covariance Functions 60
5.3.2 Predictions with the GP model 63
5.3.3 Learning a GP model 65
5.4 Efficient Regression by Problem Decomposition 66
5.5 Experimental Results 69
5.5.1 Deformation Cost Function Example 71
5.5.2 GP Training and Number of Training Samples 73
5.5.3 Number of Nearest Neighbors 74
5.5.4 Modeling Uncertainty 75
5.5.5 Statistical Evaluation 80
5.5.6 Computation Time 80
5.6 Summary 82

6 Motion Planning for Real Robots 83
6.1 The Motion Planning Framework 84
6.1.1 Path Costs 86
6.1.2 Roadmap Deformation Costs 86
6.2 Planning for Manipulators in 3D 90
6.3 Robot Navigation in 2D 93
6.4 Sensor-based Collision Avoidance for Non-deformable Objects 96
6.4.1 Learning a Sensor Model for Deformable Objects 97
6.4.2 Classifying Sensor Measurements and Avoiding Collisions 98
6.5 Experimental Results 100
6.5.1 Arm Planning in 3D 100
6.5.2 Robot Navigation in 2D 104
6.5.3 Computation Time 107
6.6 Summary 109

7 Related Work 111
7.1 Deformable Modeling and Parameter Estimation 111
7.1.1 Deformation models 111
7.1.2 Parameter estimation 112
7.2 Robot Motion Planning and Learning 115
7.2.1 Robot Motion Planning 115
7.2.2 Motion Planning with Deformable Objects 116
7.2.3 Robot Learning with Gaussian Processes 118

8 Discussion and Outlook 121
8.1 Concluding Remarks 125

Bibliography 127

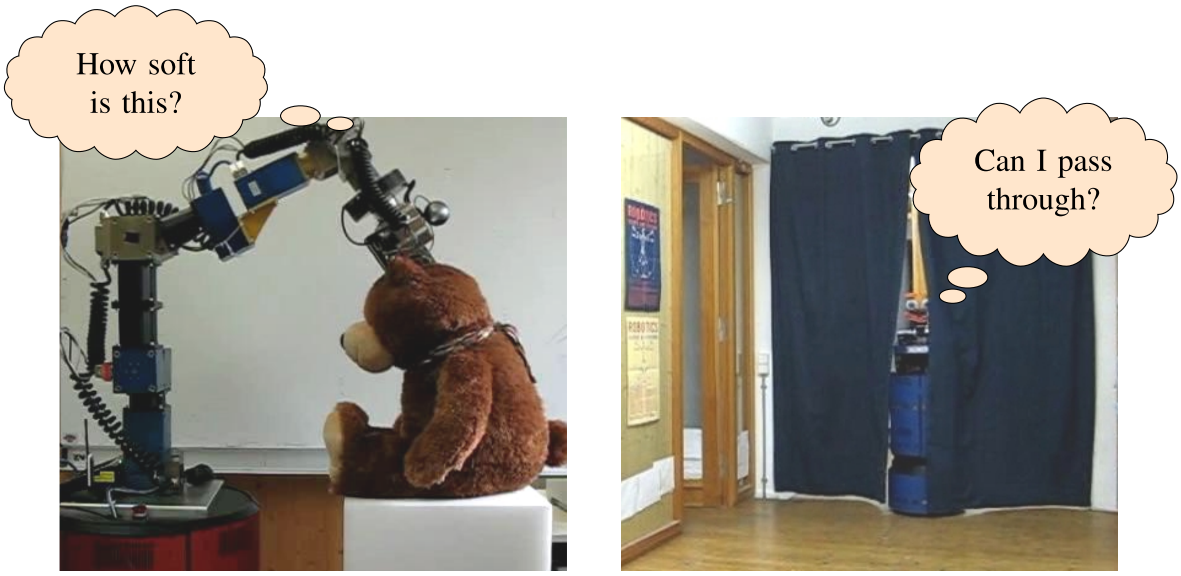
1 Introduction

Autonomous intelligent robots operating in real-world environments are required to plan their motions and reason about their actions. Planning and reasoning, in turn, require an appropriate model of the environment. Traditionally, the environment is assumed to be rigid and obstacles are to be avoided by the robot. This assumption is reasonable for many applications, and approaches have been developed to model such environments, for example using simultaneous localization and mapping (SLAM) techniques, and to plan the motions of agents therein. Applications include service robots operating in indoor environments. One successful example is the autonomous interactive museum tour-guide Rhino (Burgard et al., 2000); autonomous agents operating in urban environments include the Google driverless car (Guizzo, 2011), or the European robotic pedestrian assistant Obelix (Kümmerle et al., 2013) that successfully navigated in crowded downtown Freiburg.

In real-world environments, many objects a robot encounters are non-rigid, and considering the deformability of objects greatly broadens the robot's options to achieve goals. Consider, for instance, obstacles, such as curtains, or plants. A robot that knows that the curtain can be easily deformed or that the leaves of a plant can be gently bent away, can possibly avoid large detours when moving to a goal location. It can also be able to reach a goal that would be blocked if all objects were considered as rigid obstacles. In these examples, the robot can benefit from taking into account object deformations instead of simply avoiding these obstacles. A similar problem arises, for instance, in the context of robotic surgical assistants. Here, the deformations of tissues and organs need to be considered and their damage and deformation must be minimized when planning the motions of a surgical tool. This problem has been addressed, for instance, in the works of Alterovitz et al. (2009) and Patil et al. (2011). Beyond motion planning, other interesting applications considering the deformation properties of objects are conceivable. A robot might have to grasp deformable objects with the right amount of force, as considered in the work of Gopalakrishnan and Goldberg (2005), or bring them into desired deformed configurations. It can be required to manipulate deformable objects, for instance, when tying knots (Moll and Kavraki, 2006; Saha and Isto, 2007) or when folding laundry (Cusumano-Towner et al., 2011; Maitin-Shepard et al., 2010).

In this thesis, we study the problem of robot motion planning in environments with deformable objects. Figure 1.1 illustrates the two key questions that arise in this context. To begin with, a robot needs to be able to learn about the deformation properties of objects. Furthermore, it must be able to consider potential object deformations when planning its motions. In our example, the robot wants to choose a motion that minimizes the deformations of the curtains.

Considering deformable objects when planning a robot's actions requires an appropriate deformation model. Such a model includes the elastic properties of the material, besides the geometry of the object and can be used to perform simulations of different robot actions and to evaluate their outcome. In recent years, deformation simulations and underlying physical models have been investigated in the computer graphics community. Applications range from the animation of virtual characters and environments in movies and games, over haptic rendering to surgery simulations. Popular underlying physical deformation models include mass-spring models and finite element models based on continuum elasticity theory. One important question in this context is how to determine the elasticity parameters of these deformation models. To some extent, they can be hand-tuned such that the simulation leads to visually plausible behavior. However, this is a tedious task for a user, and in real-world applications, such as surgical simulations, an accurate deformation model is required. To determine the elasticity parameters of objects, some sort of interaction with them is required to observe their deformation behavior when subject to external forces. Different approaches that address the problem of estimating material parameters based on interaction have been presented. Kauer et al. (2002), for instance, investigated complex deformation models of biological tissues for use in surgical simulations. To acquire data of tissue deformations, they developed an aspiration instrument that is operated by a human surgeon. Robots have also been used to interact with objects and to acquire data necessary to estimate material parameters. For example, Lang et al. (2002) discuss the acquisition of deformation models of real objects with a robotic facility employing different sensors and actuators. In this thesis, we investigate how to determine appropriate models with a manipulation robot that is able to interact with objects, as indicated in Figure 1.1a. Our focus is on as autonomous a model acquisition as possible, the robot should be able to rely on its actuators and sensors to make the necessary observations for estimating the material parameters of the object.



(a) (b)

Figure 1.1: The two central questions we address in this thesis: how can a robot learn about the deformation properties of objects and how can it incorporate this knowledge when planning its motions?

Considering deformable objects increases the complexity of the planning problem, as potential object deformations introduce many new degrees of freedom. In contrast to traditional planners for static environments, the state space to be considered not only consists of independent robot configurations – object deformations depend on the path of the robot. As Figure 1.1b illustrates, the planner must reason about the question: does a path lead to object deformations and if so, how strenuous is its execution for the robot and the object? Given a deformation model of the obstacle, this question can be answered using simulations. Planning approaches following this idea have been investigated by Lamiraux and Kavraki (2001) in the context of deformable robots, or by Rodríguez et al. (2006) in the context of completely deformable environments. Computing a path to a goal typically involves evaluating many alternatives and choosing one that optimizes a cost function for the robot, considering, for instance, exertion of forces, travel length, or a trade-off between different costs. Since the corresponding simulations are time-consuming, answering such a path query requires significantly more computation time than in static environments, the approaches mentioned above report computation times of ten minutes and more for a single path query. For robots moving in real-world settings, however, a quick response to new tasks is desired. To achieve this, we investigate techniques to avoid simulations during planning time and to represent the expected costs for a path resulting from object deformations using regression.

With a model of the environment and deformable objects and a planner considering these models, the foundations for autonomous navigation in such environments are laid. Once a plan is determined, however, the robot still needs to execute the plan to reach a goal and to accomplish a task. For real robots, this poses a set of additional challenges, as this process is subject to different types of uncertainty, concerning, for instance, the actual position of the robot and the deviation from the planned trajectory, but also its model and the appearance of new, possibly dynamic obstacles.

In the context of wheeled robots navigating in rigid environments, approaches to pose estimation, localization, and collision avoidance have been presented that guarantee a safe and reliable execution of a planned trajectory. Applying them to deformable environments, however, is not straight-forward. Localization approaches, for instance, rely on the assumption that observations of the robot only depend on its actual position. If object deformations are involved, the observations in a given position also depend on past configurations. In particular, the problem of collision avoidance needs to be reconsidered: object deformations necessarily imply contact between the robot and the object. At the same time, the robot might encounter dynamic obstacles, such as humans populating the environment. Collisions with such obstacles are to be avoided. Therefore, in the context of path execution, we address the question of how to interpret the robot's sensor measurements and how to distinguish observations of deformable objects from other observations. This is required to ensure safe and efficient navigation.

These considerations can be summarized in the following research questions that we investigate in this thesis:

- How can a robot learn about the deformation properties of objects?
- How can a robot efficiently consider object deformations when planning its motions?
- How can a robot successfully navigate among deformable objects?

This thesis addresses these three research questions and provides solutions to all of them. First, it develops an approach to determine the elasticity parameters of objects and a motion planning framework that considers learned deformation models to optimize the trade-off between travel costs and object deformations. In particular, our focus is on an efficient representation of deformation costs that can be expected along a robot trajectory. Second, for the assumption of stationary deformable objects, we investigate object-specific deformation cost functions that can be precomputed and thus avoid time-consuming simulations when planning motions to a given goal. Finally, we study applications of the developed planner on different types of robots acting in real-world environments.

1.1 Contributions of this Thesis

We developed several approaches addressing individual aspects that need to be considered when realizing a navigation system for robots acting in real-world environments with deformable objects. These include environment modeling, planning, and their application to different robots. Our contributions can be summarized as follows.

Deformation model learning

We present an approach to learn deformation models of real objects with a robotic manipulator that interacts with deformable objects and is equipped with sensors to measure their deformations and the deformation forces. The material parameters of an object are determined by fitting the observations of the robot to a linear finite element model.

Deformation cost functions for planning

We present an approach to model the costs of deformations that can be expected on a trajectory using nonparametric Gaussian process regression. We assume objects to be deformable but stationary, which allows us to interpret the deformation costs associated with an object as a function of the robot trajectory. Simulations can be carried out in a preprocessing step to generate samples that are used for training the Gaussian process. The efficiency of the motion planner is increased by replacing time-consuming simulations with efficient Gaussian process evaluations.

Motion planning for real robots

We demonstrate the applicability of our developed planning system on two different robotic platforms: a robotic manipulator with seven degrees of freedom and a wheeled robot navigating in indoor environments. Furthermore, we address the problem of collision avoidance arising when wheeled robots navigate in environments populated by humans. We present an approach that enables a robot to interpret its sensor measurements and to distinguish between allowed collisions with deformable objects and collisions with dynamic obstacles based on its sensor measurements when moving along a trajectory.

1.2 Publications

The work presented in this thesis is based on the following publications given in chronological order.

- B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Efficient motion planning for manipulation robots in environments with deformable objects. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011a.
- B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Using Gaussian process regression for efficient motion planning in environments with deformable objects. In *Proc. of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*, 2011b.
- B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010a.
- B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *Proc. of the Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conference (RSS)*, 2010b.
- B. Frank, C. Stachniss, R. Schmedding, M. Teschner, and W. Burgard. Real-world robot navigation amongst deformable obstacles. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.
- B. Frank, M. Becker, C. Stachniss, M. Teschner, and W. Burgard. Efficient path planning for mobile robots in environments with deformable objects. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008a.
- B. Frank, M. Becker, C. Stachniss, M. Teschner, and W. Burgard. Learning cost functions for mobile robot navigation in environments with deformable objects. In

Workshop on Path Planning on Cost Maps at the IEEE Int. Conf. on Robotics & Automation (ICRA), 2008b.

1.3 Collaborations

This thesis was carried out within the interdisciplinary transregional collaborative research center *Spatial Cognition: Reasoning, Action, Interaction*. Fruitful collaborations within research project A2-[ThreeDSpace] with Rüdiger Schmedding and principal investigators Prof. Dr. Matthias Teschner and Prof. Dr. Wolfram Burgard resulted in several joint publications. The approach to deformation model learning (Chapter 4) and their application to robot navigation (Chapter 6) were developed in close collaboration with Rüdiger Schmedding. The approach to modeling deformation cost functions using Gaussian process regression (Chapter 5) was investigated jointly with Axel Rottmann and Cyrill Stachniss. The roadmap planner for manipulation planning (Chapter 6) was jointly developed with Nichola Abdo in the context of his student project. The motion planning approach and the sensor model for deformable objects (Chapter 6) are based on joint work with Cyrill Stachniss.

2 Overview

In this chapter, we give an overview of our approach to motion planning in environments with deformable objects and introduce the individual parts of the system that are discussed in the subsequent chapters.

First of all, the robot needs to acquire an appropriate deformation model of an obstacle that allows for simulations of the interactions with the robot. Such a model consists of a geometric description together with a set of material parameters that describe how the object deforms under external forces. Our robot is equipped with a force sensor and a depth camera; it interacts with the object and measures the surface deformations and the corresponding deformation forces. The geometric model is acquired with the depth camera and the material parameters are determined using an inverse finite element method by optimizing the deformation parameters of the finite element model such that it best fits the observations of the robot. After introducing the underlying physical model in Chapter 3, we present our approach to acquire such models with a manipulation robot in Chapter 4.

Our motion planner is based on the probabilistic roadmap framework introduced by Kavraki et al. (1996) and allows for efficient answering of multiple path queries in a static environment. In our application, the cost function of the planner optimizes the trade-off between travel and deformation costs. When generating a motion plan for the robot, the learned deformation models can be used in a finite element simulation to evaluate the costs of deforming objects for different robot trajectories. Finite element simulations, however, are time-consuming, and typically, thousands of different alternative trajectories must be evaluated when searching for a motion plan to a specific goal point.

To improve the efficiency of the planner, we present an approach to learn object-dependent deformation cost functions. We assume that the environment is static and does not change on its own over time, and that obstacles can indeed be deformed but cannot be moved by the robot. Furthermore, we only consider interactions between the robot and obstacles and

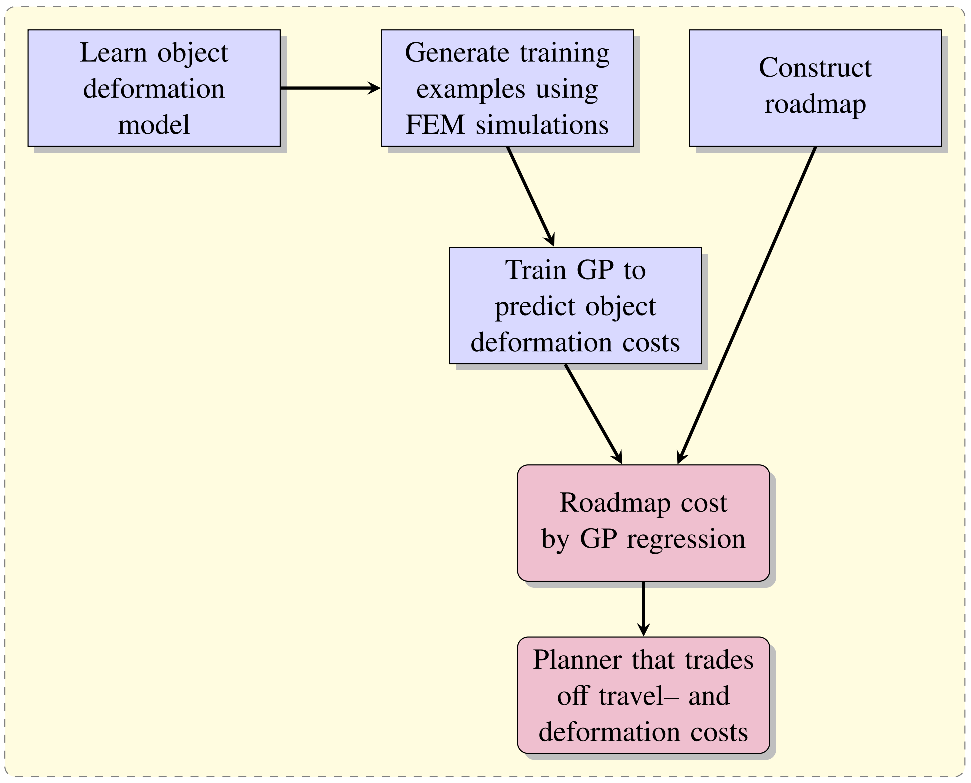

Learn object deformation model
Generate training examples using FEM simulations
Construct roadmap
Train GP to predict object deformation costs
Roadmap cost by GP regression
Planner that trades off travel– and deformation costs

Figure 2.1: Overview of our planning system: preprocessing steps to determine an appropriate model required for the planner (blue) and online steps to compute a motion plan to a specific goal (red).

neglect interactions between different obstacles. This allows us to generate a set of training examples of robot trajectories that lead to object deformations offline by carrying out corresponding finite element simulations. We model a deformation cost function for each object individually using Gaussian process (GP) regression. The samples of simulated robot trajectories are used to train a GP model and to estimate the deformation costs of new trajectories generated by the planner in an efficient way without the need for time-consuming simulations during runtime. This approach is presented in Chapter 5.

The individual steps to generate appropriate models and their interplay to obtain a working planning system are illustrated in Figure 2.1. We finally present two implementations of our planning system on a wheeled robot and a manipulator in Chapter 6.

3 Background

The basis of our planner, which considers non-rigid obstacles in the environment, is a simulation system that allows us to compute the interactions between the robot and the obstacles. The simulation system is required to simulate robot motions, determine potential collisions with obstacles that occur during its movements, compute appropriate collision forces and the resulting deformations of objects. An underlying deformation model describes how objects deform under applied forces depending on their material properties.

As a simulation environment, we use *DefCol Studio*,[1,2] which implements different methods for modeling deformable objects, handling collisions between objects, and integrating the dynamic behavior of the objects over time. In the following, we describe the parts of the simulation system that are of interest to our approach and go into details of the underlying deformation model. We make use of the simulation system not only when planning robot motions, but in a first step also when learning deformation models of real objects that the robot could encounter as obstacles in its environment.

3.1 Deformation Simulation

For the dynamic simulation of object deformations, first of all, an appropriate object representation is required. In *DefCol Studio*, objects are represented by a tetrahedral mesh. Such a volumetric mesh models the entire object including its interior, in contrast to a surface mesh that only represents the boundary of an object. The nodes of the tetrahedrons are furthermore considered as mass points of the object. Figure 3.1 illustrates this representation. Based on this geometric description, different deformation models can be used,

[1]B. Heidelberger: DefCol Studio – Interactive deformable modeling framework. `http://www.beosil.com/projects.html#DefColStudio`, last accessed November 13, 2012.

[2]M. Teschner: Defcol Studio 1.1.0. `http://cg.informatik.uni-freiburg.de/software.htm`, last accessed November 13, 2012.

for instance, mass-spring systems, in which the nodes of the tetrahedrons are connected by virtual springs (Teschner et al., 2004), meshless deformation models (Müller et al., 2005), or finite element models (Müller and Gross, 2004), which approximate a continuous deformable object by a finite set of volumetric elements, tetrahedrons in this case. We use the finite element method to model the deformation behavior of objects and go into details on how to compute deformations of an object given external forces in Section 3.2.

In the following, we discuss different aspects of the dynamic simulation of objects that are necessary to compute interactions between objects. The simulation integrates the motions of objects over time and proceeds as follows: In each time step, it computes the deformations of objects and their unconstrained motions, then it detects collisions and computes contact forces for the colliding points. Finally, it constrains the motions of the object points such that a collision-free state is maintained.

3.1.1 Collision Detection

For a realistic simulation of the interactions between the robot and deformable objects, an efficient collision detection algorithm is required. In our framework, we employ the spatial subdivision scheme of Teschner et al. (2003). The key idea of this approach is to implicitly discretize $\mathbb{R}^3$ into small uniform 3D grid cells and to map the elements contained in the grid cells to a hash table. Consequently, only elements with the same hash key need to be checked for collisions.

The collision detection algorithm proceeds by first mapping all vertices of all elements to the hash table. Second, all tetrahedrons are mapped to the hash table by inserting all grid cells that intersect with the axis-aligned bounding box of the tetrahedron. In each case, the hash key is computed from the coordinates of the corresponding grid cell. Finally, all vertices and tetrahedrons with the same hash index are tested for intersection unless a vertex is part of a tetrahedron. The actual intersection test computes barycentric coordinates of a point with respect to a tetrahedron, which directly shows whether a point lies within a tetrahedron. Thus, collisions and self-collisions can be detected. The performance of this approach depends on different parameters, such as hash table size, hash function, and grid cell size. Teschner et al. (2003) investigated and optimized these parameters and showed that this collision detection method allows for the interactive simulation of objects consisting of 20,000 tetrahedrons and 6,000 vertices at 15 Hz.

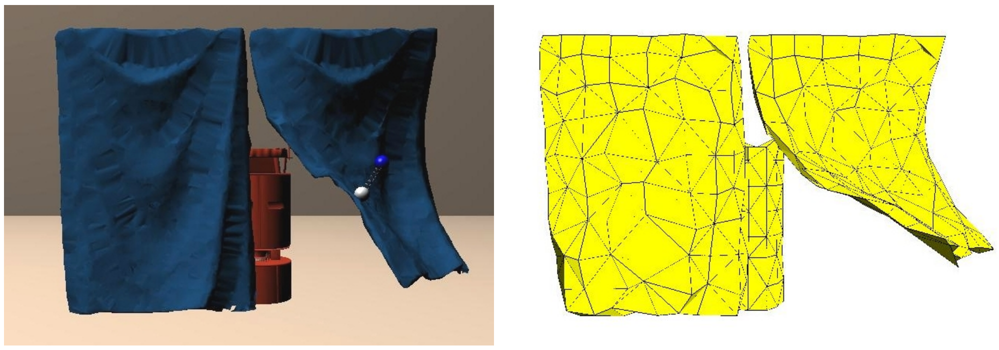


Figure 3.1: Object models in the simulation environment: a textured high-resolution surface representation is used for visualization and a discretization into a coarse volumetric mesh is used to compute the deformations.

Since space is usually filled sparsely and non-uniformly, this method requires less memory than an explicit discretization, for instance, based on octrees (Bandi and Thalmann, 1995) or binary space partitioning with BSP trees (Melax, 2000). Furthermore, the hash grid is independent of the objects in the environment, in contrast to an explicit discretization, and can be computed efficiently. An overview of collision detection in the context of deformation simulations is given by Teschner et al. (2005). Bounding volume hierarchies, for instance, based on spheres (Palmer and Grimsdale, 1995), axis-aligned bounding boxes (Van den Bergen, 1997), or oriented bounding boxes (Gottschalk et al., 1996), are an alternative to spatial subdivisions and are widely used for collision detection of rigid objects. A substantial effort is invested in precomputing the hierarchy for an object for the benefit of fast collision checks. As Teschner et al. (2005) pointed out, this advantage cannot be fully exploited in simulations of deformable objects since these objects frequently change their shape and thus, their hierarchies need to be updated or recomputed.

3.1.2 Computation of Contact Forces

Once collisions between objects are detected in the simulation, their contact needs to be handled and appropriate contact forces need to be computed. When two bodies collide and are in contact, they exert forces onto each other in the contact region. According to Newton's third law, these forces are equal in magnitude and opposite in direction. In case of rigid bodies, these forces change the momentum of the objects and accelerate them such that they move away from each other. In case of elastic objects, parts of these forces lead to deformations of the colliding objects. Spillmann et al. (2007) consider these physical principles when deriving contact forces for colliding elastic objects. Their approach is implemented in *DefCol Studio*, and we use it to handle collisions between the robot and deformable objects in a physically sound way. In the following, we briefly outline, how these contact forces are computed.

The general idea of Spillmann et al.'s collision handling scheme (2007) is to consider in each time step the unconstrained motions of objects, due to gravity, elastic forces acting inside objects, among others, and to determine the set of collisions resulting from these unconstrained motions. Then, contact forces are computed for each colliding point that obey the laws of contact dynamics and ensure an overall collision-free state. Finally, the collision-free state of all objects is maintained by applying the contact forces to the unconstrained positions of object points.

More precisely, after having computed the unconstrained motions, the set of collisions C is considered that would occur in the next time step $t + \Delta t$ as a result of these motions. A collision is represented by a point i of an object and a corresponding triangle T_i on the surface of the colliding object, which belongs to the volume the point has penetrated. The idea is to derive local forces $\mathbf{f}_i$ and $\mathbf{f}_{T_i}$ for each collision, acting on the point i and on the colliding triangle T_i. According to Newton's third law, these local forces are of equal size but opposite direction:

$$\mathbf{f}_i + \mathbf{f}_{T_i} = 0. \tag{3.1}$$

Each triangle is represented by points j, k, l, which in turn are involved in collisions with other triangles. Similar, each point can be considered as belonging to several triangles that are involved in collisions. Thus, the total force acting on a point is given as $\mathbf{F}_i = \sum_{i \in C} \mathbf{f}_i + \sum_{T_j \in C : i \in T_j} \mathbf{f}_{T_j}$, the sum of forces $\mathbf{f}_i$, resulting from collisions with triangles, and the sum of forces $\mathbf{f}_{T_j}$, where i is involved in the collision as part of a triangle T_j. A global force equilibrium is enforced to guarantee conservation of energy:

$$\sum_i \mathbf{F}_i = 0. \tag{3.2}$$

The contact forces are related to the penetration depths of points. Thus, for each collision, a penetration depth vector $\mathbf{d}_i^{t+\Delta t}$ is computed using the approach of Heidelberger et al. (2004). Figure 3.2 illustrates this idea. The penetration depth of the colliding point is assumed to correspond to those of the triangle points, although this is not the case in general.

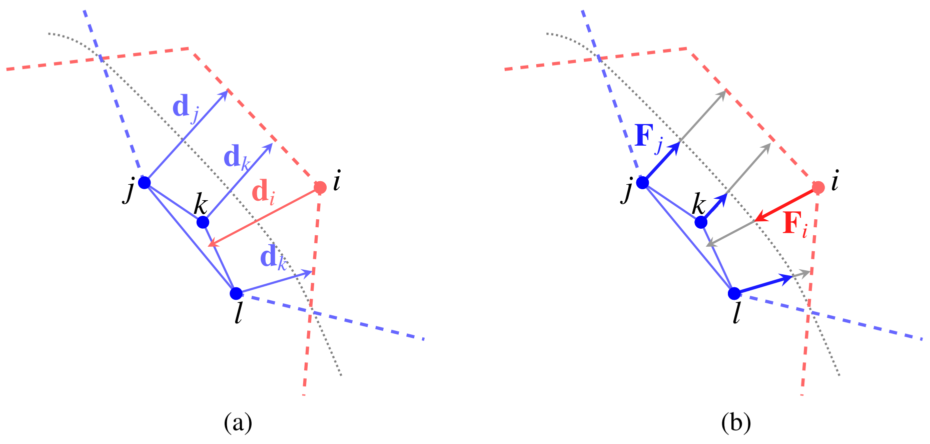


Figure 3.2: Contact force computation: we consider point collisions consisting of a point i belonging to the red object and interpenetrated triangle (j, k, l) belonging to the blue object. (a) For all colliding points, a penetration depth vector $\mathbf{d}$ is computed. (b) Local forces are computed that accelerate points onto the virtual contact surface (represented by the gray line).

With this assumption, the computation of a contact force $\mathbf{F}_i$ for a colliding point i with unconstrained position $\tilde{\mathbf{x}}_i^{t+\Delta t}$ and mass m_i reduces to finding a scalar α_i that accelerates it onto the virtual surface of contact in the next time step Δt (see Figure 3.2b):

$$\mathbf{x}_i^{t+\Delta t} = \tilde{\mathbf{x}}_i^{t+\Delta t} + \alpha_i \mathbf{d}_i^{t+\Delta t}. \tag{3.3}$$

The local forces acting on a point and the colliding triangle can be derived by incorporating Newton's law of motion $\mathbf{f} = m \cdot \mathbf{a}$ in Eq. (3.3) – which will be discussed in detail in the next section – furthermore, by enforcing the constraints stated in Eq. (3.1) and (3.2):

$$\mathbf{f}_i = c_i \frac{m_i}{\Delta t^2} \mathbf{d}_i^{t+\Delta t} \alpha_i \qquad \text{and} \qquad \mathbf{f}_{T_i} = \sum_{j \in T_i} c_i h_{ij} \frac{m_j}{\Delta t^2} (-\mathbf{d}_i^{t+\Delta t})(1 - \alpha_i). \tag{3.4}$$

Here, the elements h_{ij} provide the barycentric coordinates of a contact point i with respect to a point j of the penetrated surface triangle T_i, and thus implicitly consider the unconstrained position of a point. The constants

$$c_i = \frac{1}{1 + \sum_{j=1}^{n} h_{ji}}, \tag{3.5}$$

can be considered as weights that take into account the barycentric coordinates h_{ji} of all points j that are in contact with a triangle containing the point i. These weights consider the contribution of each mass point to the local force. The α_i can be derived by enforcing Eq. (3.1) for Eq. (3.4) and solving for α_i. This results in

$$\alpha_i = \frac{\sum_{j=1}^{n} c_j h_{ij} m_j}{c_i m_i + \sum_{j=1}^{n} c_j h_{ij} m_j}, \tag{3.6}$$

which can be computed independently for each contact point. The total force $\mathbf{F}_i$ acting on a point as the sum of all local forces then amounts to

$$\mathbf{F}_i = \frac{m_i}{\Delta t^2} \mathbf{d}_i^{t+\Delta t} \alpha_i. \tag{3.7}$$

This contact force $\mathbf{F}_i$ is exactly the sum of all local forces $\mathbf{f}_i$ and $\mathbf{f}_{T_j}$ acting on a point i involved in collisions either as contact point or as part of a colliding triangle T_j. For further details regarding the derivation of the contact forces, the treatment of border cases and the correction of the approximation error that is introduced by assuming equal penetration depths for colliding points and triangles, we refer to the paper of Spillmann et al. (2007).

As we will see in the next section, in which we address the integration of the movement of points over time, application of the proposed contact forces immediately results in the constrained collision-free position for each point. The contact forces thus depend on the positions of the points, the internal forces acting on the points, and the impulses generated by the collisions.

In principle, there are two different approaches to collision handling in simulation: penalty methods and constraint methods. In general, when collisions are detected, objects already partially overlap and interpenetrate each other. Penalty-based collision handling approaches, such as the approach presented by Heidelberger et al. (2004), compute response forces according to an interpenetration measure for the colliding points. This is computationally efficient but leads to physically implausible results as it usually takes several simulation steps to resolve a collision. In addition, the definition of a stiffness constant is required; this constant must be large enough to resolve collisions but at the same time small enough to avoid overshooting and instabilities, thus it is difficult to determine an appropriate constant. Constraint-based approaches, in contrast, impose non-penetration constraints derived from contact dynamics on the colliding points and compute the response forces from these constraints (e. g., Baraff and Witkin, 1992). The computation of contact forces can be formulated as linear complementary problem and solved iteratively, which unfortunately is expensive to compute and can be subject to numerical inaccuracies. The collision-handling scheme of Spillmann et al. (2007) avoids iterative computations and thus combines the advantages of both penalty and constraint-based approaches. It is furthermore able to deal with different deformation models, such as mass-spring models, linear, and nonlinear finite element models. The complexity of the approach is linear in the number of collisions. Experiments suggest that simulations can still be carried out with 2 Hz for scenes with up to 80,000 tetrahedrons.

3.1.3 Time Integration

To simulate the dynamic behavior objects over time, Newton's equation of motion $\mathbf{f} = m\mathbf{a}$ is integrated for each mass point $\mathbf{x}$ with mass m and force $\mathbf{f}$ acting on it. The force $\mathbf{f}$ partially results from contacts between objects, as discussed above, partially it results from internal forces acting inside the object to restore its undeformed state. We will discuss these internal forces in more detail in the next section. With these internal and external forces available, the simulation solves the following second order partial differential equation:

$$\mathbf{f} = m\ddot{\mathbf{x}}. \tag{3.8}$$

This equation is discretized into small time steps Δt and solved numerically using the Verlet integration scheme (Verlet, 1967): $\ddot{\mathbf{x}}$ at time step t is approximated as

$$\begin{aligned}\ddot{\mathbf{x}}(t) &= \frac{\dot{\mathbf{x}}(t+\Delta t) - \dot{\mathbf{x}}(t)}{\Delta t} \\ &= \frac{1}{\Delta t}\left(\frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t} - \frac{\mathbf{x}(t) - \mathbf{x}(t-\Delta t)}{\Delta t}\right) \\ &= \frac{\mathbf{x}(t+\Delta t) - 2\mathbf{x}(t) + \mathbf{x}(t-\Delta t)}{\Delta t^2}.\end{aligned} \tag{3.9}$$

Inserting Eq. (3.9) into Eq. (3.8) results in the equation to compute subsequent positions of a point $\mathbf{x}$ as

$$\mathbf{x}(t+\Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t-\Delta t) + \Delta t^2 \frac{\mathbf{f}}{m}. \tag{3.10}$$

This integration scheme is frequently used in the simulation of deformable bodies and provides a good trade-off between accuracy and efficiency.

Furthermore, it is used in the derivation of the contact forces in the previous section to ensure that their application immediately results in constrained collision-free positions. This can be shown by integrating the sum of the forces acting on a point, the internal force $\mathbf{p}_i$ and the external force $\mathbf{F}_i$ from Eq. (3.7):

$$\begin{aligned}\mathbf{x}_i(t+\Delta t) &= 2\mathbf{x}_i(t) - \mathbf{x}_i(t-\Delta t) + \frac{\Delta t^2}{m_i}(\mathbf{p}_i + \mathbf{F}_i)\\ &= 2\mathbf{x}_i(t+\Delta t) - \mathbf{x}_i(t-\Delta t) + \frac{\Delta t^2}{m_i}\mathbf{p}_i + \frac{\Delta t^2}{m_i}\mathbf{F}_i\\ &= \tilde{\mathbf{x}}_i(t+\Delta t) + \frac{\Delta t^2}{m_i}\mathbf{F}_i. \end{aligned} \tag{3.11}$$

Thus, the constrained position $\mathbf{x}_i$ at time step $t + \Delta t$ is exactly as required by Eq. (3.3), namely collision-free and on the contact surface.

3.2 Deformation Model

So far, we have discussed how the simulation framework proceeds to compute the dynamic behavior of deformable objects and their interactions. The only open question in this context is the underlying deformation model, which determines how an object deforms when subject to external forces.

In contrast to a rigid object, an elastically deformable object can be deformed by external forces; furthermore, it returns to its original shape after the external forces leading to a deformation are removed. This property is due to internal forces that act inside the elastic object to counteract the external forces and to restore its undeformed state. The deformation model we use is based on continuum mechanics theory (Chung, 1996), we consider the distribution of forces and energies inside a solid object. Formally, a deformable solid object can be described by its undeformed state, given by a set of points $\mathbf{x} = (x, y, z)$ belonging to the object and a set of material parameters that determines how the object deforms under external forces. A deformation is then specified by a displacement field $\mathbf{u}(\mathbf{x}) = (u, v, w)$, which maps each point $\mathbf{x}$ of the object in its reference position to a deformed position $\mathbf{x} + \mathbf{u}$.

The deformed state of an object is characterized by the physical quantities stress and strain. While the stress measures the forces acting inside the object to restore its original shape per unit area, the strain measures the deformation with respect to the undeformed state. The relation between stress and strain is called the *constitutive equation*. For linearly elastic, homogeneous and isotropic material, we assume a linear relation between stress $\boldsymbol{\sigma}$ and strain $\boldsymbol{\varepsilon}$ governed by the generalized Hooke's law:

$$\boldsymbol{\sigma} = C\boldsymbol{\varepsilon}. \tag{3.12}$$

To compute the distribution of elastic forces inside a continuous solid object and to establish the relation between object deformation, specified by a displacement field u, and external forces acting on the object, we consider the total potential energy Π of a solid, which is given by

$$\Pi = \Lambda + WP. \tag{3.13}$$

The inner or elastic energy Λ is the energy stored inside the object as deformation and is given as

$$\Lambda = \frac{1}{2}\int_V \boldsymbol{\sigma}^T \boldsymbol{\varepsilon}\, dV. \tag{3.14}$$

The work potential WP is determined by the external forces acting on an object:

$$WP = -\int_v \mathbf{u}^T \mathbf{f}_V dV - \int_S \mathbf{u}^T \mathbf{T}_S dS - \sum_i \mathbf{u}_i^T \mathbf{f}_i. \tag{3.15}$$

It consists of forces $\mathbf{f}_V$ distributed over the body, for instance, gravitational forces, forces distributed over the surface $\mathbf{T}_S$, and point loads $\mathbf{f}_i$ resulting from collisions with other objects. In the simulation, however, only the last term that contains the point loads is considered. A stable equilibrium configuration of a deformation can be found by minimizing the potential energy, which is done by setting the derivatives to zero and solving for the resulting displacements. This, in fact, requires solving a continuous partial differential equation, and we will shortly go into detail on how to solve this numerically using the finite element method.

3.2.1 Elasticity Parameters

In our deformation model, we assume linearly elastic and isotropic material. In this case, the matrix C governing the relation between stress $\boldsymbol{\sigma}$ and strain $\boldsymbol{\varepsilon}$ in Eq. (3.12) depends only on two independent elasticity parameters, Young's modulus E and Poisson's ratio ν (for a derivation, see Chung, 1996, Chapter 4):

$$C = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5-\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5-\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5-\nu \end{pmatrix}. \tag{3.16}$$

These parameters characterize the deformation behavior of an object and can be interpreted as follows. The Young modulus describes the stiffness of an object. It measures the force that is needed to enlarge or compress an object by some fixed amount and is given by the ratio of stress to strain in the direction of the applied force:

$$E = \frac{\sigma}{\varepsilon} = \frac{F/A}{\Delta x/x} = \frac{Fx}{A\Delta x}. \tag{3.17}$$

Its unit is force per area and it is frequently specified in $\frac{\text{N}}{\text{dm}^2}$. This is visualized in the example in Figure 3.3: a bar elongates by an amount ΔL in the direction of the applied force that is proportional to F, L and $\frac{1}{A}$. The constant of proportionality is the Young modulus.

In contrast to the Young modulus, the Poisson ratio is related to the compressibility of an object. When a material is expanded in one direction, a compression in the other two directions perpendicular to the expansion can be observed, and vice versa a compression in one direction leads to an extension in the other two directions. The Poisson ratio is thus given by the negative ratio of the transverse strain to the axial strain:

$$\nu = -\frac{\varepsilon_{\text{trans}}}{\varepsilon_{\text{axial}}} = -\frac{\varepsilon_y}{\varepsilon_x} = -\frac{\varepsilon_z}{\varepsilon_x}. \tag{3.18}$$

Since we consider isotropic material, the changes in the two directions y, z perpendicular to the direction x of the applied force are equal. In Figure 3.3, this effect is illustrated as a contraction Δd of the bar in the directions perpendicular to the applied force.

The Young modulus is always greater than zero, but not upper-bounded, with larger values characterizing stiffer materials. For isotropic objects, it can be shown that the Poisson ratio lies in the range of 0 to 0.5. A Poisson ratio of 0.5 implies perfect volume conservation, while a Poisson ratio of 0 corresponds to no volume conservation at all. For many materials including foam, it ranges from 0.25 to 0.35, for rubber it is close to 0.5.

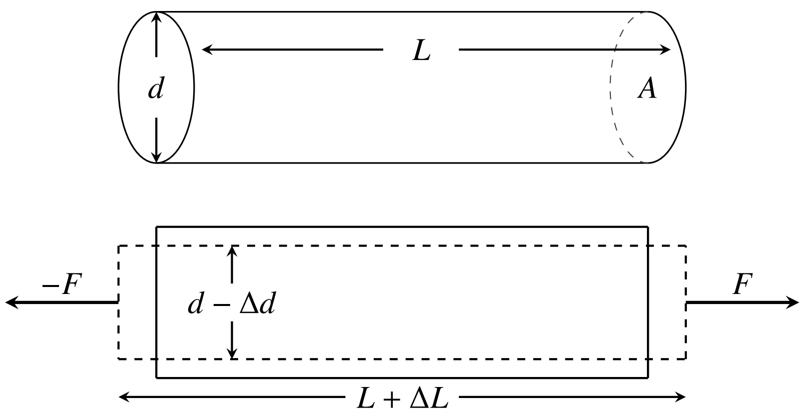


Figure 3.3: Illustration of the material parameters: a force F is applied to a cross-sectional area A of a bar with length L and diameter d. The Young modulus determines the change in the length ΔL of the bar in the direction of the applied force while the Poisson ratio determines the change in the thickness Δd in the directions perpendicular to the applied force.

3.2.2 Linear Finite Element Approximation

We use the finite element method (FEM) to approximate the deformation of a continuous object. The description we state here is based on Müller and Gross (2004) and Becker and Teschner (2007), and a more detailed derivation of the method can be found in textbooks, for instance, from Bathe (1996) or Chandrupatla and Belegundu (2002). The key idea is to discretize the object into a finite set of volumetric primitives, which are the tetrahedrons in our case, and to compute the deformations inside the elements by an interpolation of the nodal values. We recall that a deformation is specified by a displacement field **u**. The strain of an object can be expressed in terms of the gradient of its displacement field:

$$\varepsilon = \frac{1}{2}\left(\nabla\mathbf{u}^T + \nabla\mathbf{u} + \nabla\mathbf{u}^T\nabla\mathbf{u}\right), \tag{3.19}$$

with $\varepsilon \in \mathbb{R}^{3\times 3}$ the nonlinear Green-St.-Venant stress tensor. Its linearization, the Cauchy-Green strain tensor, which we also use, is given as

$$\varepsilon = \frac{1}{2}\left(\nabla\mathbf{u}^T + \nabla\mathbf{u}\right). \tag{3.20}$$

This tensor is a popular choice in computer graphics applications, since it can be computed efficiently, in contrast to the nonlinear tensor (see, e. g., Nealen et al., 2006). In the following, we use the vector notation $\boldsymbol{\varepsilon}$, which contains its six independent components and is given as:

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{zx} \\ \gamma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial y} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{pmatrix}. \tag{3.21}$$

In the finite element method, first of all, the displacement field inside a tetrahedron is approximated by the displacements of the nodes, using linear shape functions, the barycentric coordinates of the points. This can be written as a matrix multiplication

$$\mathbf{u} = N\mathbf{q}, \tag{3.22}$$

with $\mathbf{q} \in \mathbb{R}^{12}$ collecting the displacements of the nodes of the tetrahedron in a vector, $N \in \mathbb{R}^{12\times3}$ containing the shape functions. The partial derivatives of Eq. (3.20) can be determined by applying the chain rule to Eq. (3.22), (see, e. g., Becker and Teschner (2007) for a derivation) and allow us to express the strain in terms of the displacements of the nodes:

$$\boldsymbol{\varepsilon} = B\mathbf{q}, \tag{3.23}$$

where $B \in \mathbb{R}^{6\times12}$ is a constant matrix that can be precomputed for each element.

The next step of the finite element approximation is to express the equilibrium equation for the object in terms of the element displacements. Therefore, we consider the elastic energy of an element given in Eq. (3.14). We use the fact that the stress linearly depends on the strain stated in Eq. (3.12) and insert the expression for the strain in terms of the nodes given in Eq. (3.23). Thus, we can write the elastic energy Λ_e of a tetrahedral element e as

$$\begin{aligned} \Lambda_e &= \frac{1}{2} \int_e \boldsymbol{\varepsilon}^T C^T \boldsymbol{\varepsilon}\, dV \\ &= \frac{1}{2} \int_e \mathbf{q}^T B^T C^T B\mathbf{q} dV. \end{aligned}$$

Since we use linear shape functions and therefore assume the strain to be constant over an element, this simplifies to

$$\begin{aligned}\Lambda_e &= \frac{1}{2}\mathbf{q}^T B^T C^T B\mathbf{q} \int_e dV \\ &= \frac{1}{2} V_e \mathbf{q}^T B^T C^T B\mathbf{q},\end{aligned}$$

with V_e denoting the volume of e. We define the element stiffness matrix to be $K_e := V_e B^T C^T B \in \mathbb{R}^{12\times 12}$ and obtain

$$\Lambda_e = \frac{1}{2}\mathbf{q}^T K_e \mathbf{q}\,. \tag{3.24}$$

To find an equilibrium configuration of the deformable object, we minimize the total potential energy by setting the partial derivatives of $\Pi = \Lambda - WP$ with respect to the displacements $\mathbf{q}_i$ to zero.

The derivatives of Λ_e with respect to $\mathbf{q}_i$ result in

$$\frac{\partial \Lambda_e}{\partial \mathbf{q}_i} = (K_e \cdot \mathbf{q})_i, \tag{3.25}$$

and describe the elastic forces $\mathbf{f}^{\text{int}}$ acting on the nodes of the model. For the work potential WP, we only consider the external point loads $\mathbf{f}_i^{\text{ext}}$, thus the partial derivatives with respect to the displacements q_i are given by

$$\frac{\partial WP}{\partial \mathbf{q}_i} = -\mathbf{f}_i^{\text{ext}}. \tag{3.26}$$

Therefore, setting the derivative of the potential energy to zero leads to

$$\frac{\partial \Pi}{\partial \mathbf{q}} = \mathbf{K}_e \cdot \mathbf{q} - \mathbf{f}^{\text{ext}} = 0. \tag{3.27}$$

In a static equilibrium state the internal forces $\mathbf{f}^{\text{int}}$ are equal to the external forces $\mathbf{f}^{\text{ext}}$. The force displacement relation can thus be written as

$$\mathbf{f}^{\text{ext}} = K\mathbf{q}, \tag{3.28}$$

When collecting all element stiffness matrices in a global stiffness matrix K and correspondingly all displacement vectors in a global displacement vector $\mathbf{Q}$, the global force-displacement relation yields a system of equations for the tetrahedral mesh

$$\mathbf{F}^{\text{ext}} = K\mathbf{Q}, \tag{3.29}$$

that can be used to compute the deformation of an object given a force. We implicitly consider this case in Chapter 4, in which we observe a force and want to compute the corresponding object deformation for a given stiffness matrix.

For dynamic simulations, the system dynamics are obtained by solving the following partial differential equation:

$$M\ddot{\mathbf{Q}} + D\dot{\mathbf{Q}} + K\mathbf{Q} = \mathbf{F}, \tag{3.30}$$

where M is the mass matrix and D is the damping matrix. This reminds us of the Newton's equation of motion with an additional damping term $D\dot{\mathbf{Q}}$ and can be solved for the positions of the mesh using the Verlet time integration scheme discussed above.

Using linear shape functions and the linear Cauchy-Green tensor for the computation of the strain leads to problems, as the linearization assumption is only valid close to the equilibrium, that is for small deformations. Furthermore, this tensor is not invariant to rotations. This leads to ghost forces, which result in distortions for large rotational deformations. To account for that, the actual implementation in *DefCol Studio* uses the corotational finite element formulation of Hauth and Strasser (2004) and Müller and Gross (2004). The idea of this approach is to keep track of the rigid body motion for each element. This is done by extracting the rotation from the transformation matrix using polar decomposition. The forces are then computed in the back-rotated frame

$$\mathbf{f}^{\text{ext}} = RK(R^T\mathbf{q} - \mathbf{x}). \tag{3.31}$$

This formulation applies the strain tensor in the back-rotated frame, which leads to rotational invariance and in contrast to the nonlinear strain tensor to low computational costs.

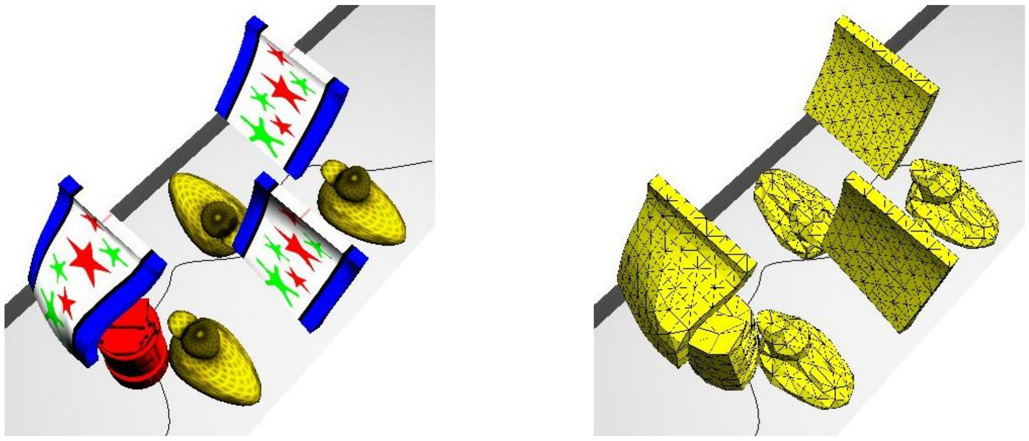


Figure 3.4: Robot navigation among deformable objects in the simulation framework discussed in this chapter: this example scene contains 3,390 tetrahedrons and can be updated with 17 Hz.

3.3 Summary

In this chapter, we discussed the framework for deformation simulations and its individual components required to simulate the interactions between deformable objects. Further, we explained the deformation model underlying the simulation that is based on elasticity theory. We are interested in the deformation model for two reasons: first, we want to determine the elasticity parameters E and ν of deformable objects. In the next chapter, we present an approach to learn these parameters from real data acquired with a mobile robot when interacting with objects. Second, with available deformation models of objects in the robot's environment, we want to perform simulations to determine the costs of robot trajectories that potentially lead to object deformations. A measure for such *deformation costs* is defined in Chapter 5 and the resulting deformation cost functions are employed in a motion planner in Chapter 6 to trade off travel costs and deformation costs for the robot. We conclude with a simulation example in Figure 3.4, which demonstrates an application of the discussed framework to simulate robot motions in an environment with deformable objects. The environment contains six deformable objects, rubber ducks and curtains. In total, this example scene contains 3,390 tetrahedrons and can be simulated with an average of 17 Hz including collision handling and FEM deformation computations.

4 Learning Deformation Models

Modeling the deformation behavior of real objects not only requires observations but also interactions with the object under investigation. In this chapter, we present our robotic system that is able to acquire data of deformable objects and to interact with them while measuring the forces and the resulting deformations.

The deformation model we want to determine consists of a three-dimensional geometrical representation and a set of material parameters that describe its elastic behavior. We first explain how to acquire geometric models of objects with our robot and how to generate volumetric meshes from these object models for application in finite element simulations. These geometric and volumetric models are a prerequisite for our approach to determine the material parameters of objects. Furthermore, we describe how the robot interacts with deformable objects in order to obtain data of object deformations that allow for the estimation of the material parameters.

In the second part of this chapter, we introduce our approach to estimate the material parameters of real objects based on an inverse finite element method. The key idea of our parameter estimation approach is to compare the observations of the robot to a finite element model and to optimize the parameters of the model. To achieve this, we use the acquired model of the object and deform it in simulation by applying the measured force to it. The parameters of the simulation, namely Young's modulus and Poisson's ratio are refined in an iterative update scheme such that the error between observed and simulated deformation is minimized.

Since our deformation model is restricted to linearly elastic, isotropic and homogeneous materials, we discuss limitations of this deformation model and investigate, to what extent these assumptions apply to the objects we consider. We present extensive evaluations of our parameter estimation approach on real and simulated data sets.

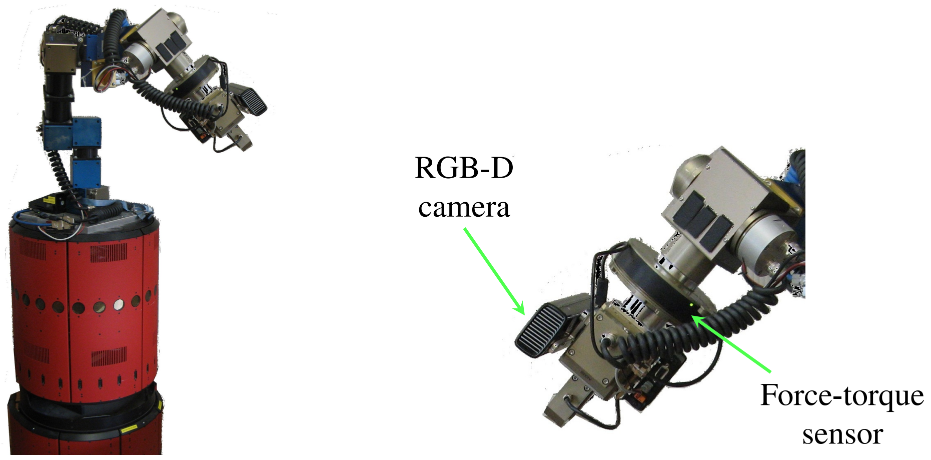


Figure 4.1: Our robotic platform for the acquisition of deformation models: the manipulation robot Zora (left) can interact with deformable objects with its manipulator. It is equipped with different sensors that are integrated into its hand (right) to observe the behavior of objects during interaction.

4.1 Data Acquisition

For the acquisition of deformation models, we set up a robotic system that is able to observe real objects and to interact with them. We present the platform in the next section before describing how to generate geometric three-dimensional models that can be used in simulations, and how to acquire data of object deformations by performing indentation tests with the robot.

4.1.1 The Robotic System

Our system for acquiring data of deformable objects consists of a mobile platform and a manipulator with seven degrees of freedom. The manipulator is equipped with a force-torque sensor and an RGB-D camera. This setup allows us to observe objects from different view points, to acquire point clouds of their surfaces, to deform objects, and to measure the corresponding deformation forces in a flexible way. The platform is illustrated in Figure 4.1.

The manipulator is built of five rotational Schunk Powercube modules and a Schunk hand with two degrees of freedom. These modules have a repeat accuracy of 0.02 degrees and therefore allow for an accurate estimation of the robot's position. We measure the deformation forces with a Schunk-FTCL-050 force-torque sensor integrated into the hand. This sensor is able to measure forces up to 300 N and torques up to 7 Nm in all three degrees of freedom. To perceive the object, we employ a Microsoft Kinect RGB-D camera, which uses the structured-light measurement principle to obtain 3D measurements of the environment. This camera is attached to the robot hand. To transform the acquired point clouds from the local camera coordinate system to the global coordinate system of the robot, we performed a hand-eye calibration using an open-source camera calibration toolbox.[1,2]

Our robot is operated using the Zora framework,[3] which includes hardware drivers, joint control, and position control using an inverse kinematics solver.[4] We implemented routines to observe objects from different viewpoints and to interact with them in a semi-autonomous way within this framework.

4.1.2 Geometric Models for Simulation

The finite element simulation described in the previous chapter requires a volumetric model of an object, in our case a tetrahedral mesh. Such a volumetric mesh can be computed from a surface mesh, which in turn is constructed from multiple scans of the object. Figure 4.2 illustrates the individual steps of the object reconstruction process, resulting in a three-dimensional geometric model. In the following, we will describe them in more detail.

Surface meshes

To acquire a three-dimensional surface representation of an object, the robot observes it from different viewpoints and records point clouds of the object. We place the object in front of the robot and manually define a set of viewpoints to achieve a good coverage of the object within the reachable workspace of the manipulator. Figures 4.2a and 4.2b show this setup and an example observation.

The recorded point clouds are registered into a consistent model. This is necessary to elim-

[1]J.-Y. Bouguet: Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, last accessed December 28, 2012.

[2]C. Wengert: Fully automatic camera and hand to eye calibration: `http://www.vision.ee.ethz.ch/software/calibration_toolbox//calibration_toolbox.php`, last accessed December 28, 2012.

[3]J. Sturm: Zora Software Package. `http://www.informatik.uni-freiburg.de/~sturm/zora-main.html`, last accessed January 7, 2013.

[4]Orocos Kinematics and Dynamics Library: `http://www.orocos.org/kdl`, last accessed January 15, 2013.

inate small errors, for instance, in the robot's pose estimate, or the hand-eye calibration, which accumulate over time and with multiple scans. The task of a registration algorithm therefore is to compute relative translations and rotations that align the surfaces correctly. In our approach, we apply a variant of the iterative closest point (ICP) algorithm proposed by Besl and McKay (1992). The basic problem formulation of scan registration is stated as follows: given two corresponding point sets, a model $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_x}\}$ and a data set $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_{N_p}\}$, we search for a transformation consisting of a rotation matrix R and translation vector $\mathbf{t}$ that aligns the data set with the model and minimizes the error

$$\mathrm{Err}(R, \mathbf{t}) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||\mathbf{x}_i - R\mathbf{p}_i - \mathbf{t}||^2 \tag{4.1}$$

for corresponding points. If the correspondences are known, this transformation can be computed in closed form, for instance, using unit quaternions (Horn, 1987) or a singular value decomposition (Arun et al., 1987).

Since the correspondences are not known in general, the ICP algorithm computes this transformation in an iterative fashion: it determines point correspondences, then computes a transformation that aligns the scans for these correspondences, applies the transformation, and evaluates the resulting error. This procedure is iterated until convergence, that is until a given accuracy is reached and $||\mathrm{Err}_i - \mathrm{Err}_{i-1}|| < \tau$ for an accuracy threshold τ. It can be shown that it monotonically converges to a local minimum (Besl and McKay, 1992) and typically yields an accurate alignment if an appropriate initial configuration is chosen.

One question left open in the above outline of the ICP algorithm is how to determine correspondences between points in different scans. Rusinkiewicz and Levoy (2001) discuss and compare different strategies to determine data associations, in terms of selecting and matching points from two scans, and weighting the point correspondences. First of all, a set of points is selected, this can be, for instance, the set of all points of one or both meshes, as in the basic ICP formulation, a randomly or uniformly sampled subset, or a feature-based selection of points based on intensities, for example. In the next step, a data association is determined for the selected points from two scans. The selected points from one scan can be paired, for instance, with the closest point in terms of the point-to-point distance or with the closest point along the surface normal in the other mesh. Finally, point correspondences can be weighted, for instance, based on their distances, to contribute with different weights to the error function. To compute the alignments for our scans, we have adopted a variant

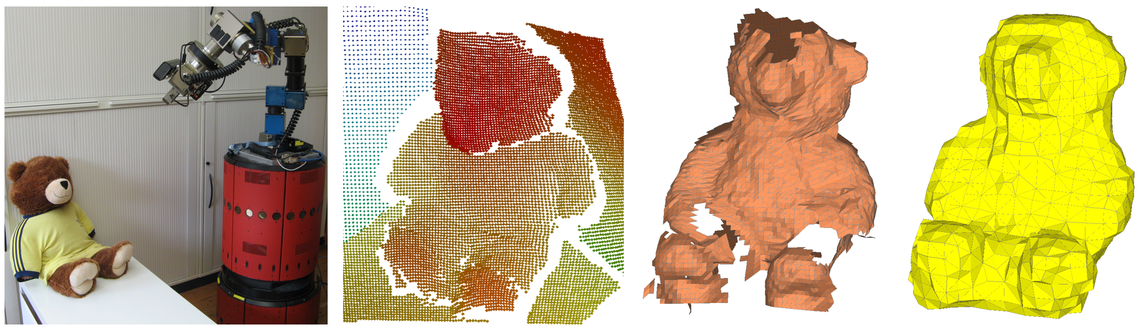

(a) (b) (c) (d)

Figure 4.2: Object reconstruction: (a) the robot observes a deformable teddy bear, (b) a point cloud obtained with the RGB-D camera, (c) the surface mesh constructed from four different point clouds, and (d) the tetrahedral mesh computed from the surface mesh. For better visibility with the RGB-D camera, the teddy is dressed with a t-shirt since its fur reflects the projected structured light pattern poorly.

called Trimmed ICP (Chetverikov et al., 2002), which uses only a fraction of the point correspondences to compute the transformations. We determine point correspondences using a nearest-neighbor data association for all points of the data set P. Then, the point correspondences are sorted according to their point-to-point distance and only a fraction of point correspondences with the smallest distances are used to evaluate the error and to compute the subsequent transformations. This approach is able to deal with only partially overlapping scans, furthermore with noise and outliers. In our experiments, we found that using the 50 % best point correspondences already leads to good alignment results.

Although convergence can be proven, all ICP algorithms are local optimization procedures and do not necessarily result in a globally optimal solution. They rely on the assumption that the scans are already roughly aligned initially. In our system, we can derive a good initial alignment from the position estimate of the manipulator, to which the camera is attached. The quality of the position estimate depends on the accuracy of the encoders in the manipulator, which is around 0.02 degrees per joint. For the kinematic structure of the robot, this adds up to an error of $\pm$ 1 mm in the position of the end effector. The position estimate is further corrupted by noise, resulting from small movements of the mobile platform due to impacts of the manipulator movements. Nevertheless, it is still sufficiently small to allow the ICP algorithm to converge to an accurate alignment.

We arrive at a global model of an object by consecutively registering new scans to the model already constructed. To account for errors that accumulate with multiple scans and correct them when closing a loop, global optimization techniques can be used, as suggested, for instance, by Pulli (1999), or Grisetti et al. (2010). In our current framework, however, we have not implemented such a global optimization. The completed model of an object can be used in simulations and represents obstacles in the environment of the robot.

Some practical issues require consideration when estimating the material properties of an object. Consider, for example, the plush teddy bear: this object changes its shape not only due to deformations but also due to "articulated" body parts, its legs and arms. When performing indentation tests with this object to measure its elastic behavior, we might want to place it on a table, lying, to probe its belly. Since we want to compare the deformations simulated on the model with the actual observations of the robot, we have to construct a model that represents the object in this configuration. Another issue is the problem of occlusion, it is not always possible for the robot to observe an object from all necessary viewpoints to obtain a complete mesh. This might be the case, for instance, if the object is sitting in front of a wall or lying on a table, or in case the robot's workspace does not allow it to move to appropriate view points. An approximately closed mesh, however, is required to compute a volumetric mesh in the next step. To obtain a closed surface mesh that clearly limits the object, we complete the model by assuming a planar surface for the unobserved parts. These planar surfaces can be extracted, for instance, from the walls or the table surface. In this way, we can generate a model for the parameter estimation almost from scratch without much overhead for exploration. Thus, the robot can immediately start to interact with the object and to acquire deformation data. Our experiments show that a complete model is not needed to estimate the deformation parameters – a partial model is sufficient. For instance, the legs of the teddy bear are not of particular interest if we consider deformations in its belly. Nevertheless, the elasticity parameters estimated in this way can be transferred to complete geometric models, given the assumption of homogeneous material.

To obtain a representation of the object's boundary, we finally generate a surface mesh from the point cloud model. In our application, we can easily compute a triangulation of the individual scans by connecting points corresponding to neighboring pixels of the RGB-D camera with a distance smaller than a threshold, in our implementation this threshold was set to 10 cm. The resulting surface mesh of the object is used in the next step to determine the volumetric tetrahedral mesh. An example surface mesh is illustrated in Figure 4.2c.

Tetrahedral meshes

To construct a tetrahedral mesh of an object, we use the meshing approach of Spillmann et al. (2006). This approach first computes a signed distance field from the surface mesh, in which voxels having a negative sign represent the volume occupied by the object. In a second step, the spatial domain described by the distance field is divided into a uniform axis-aligned grid limited by the bounding box of the object. All cells in this grid that contain no voxel with negative sign are discarded. The remaining cells are an approximation of the object's volume; the quality of this volume approximation is determined by the grid resolution. The grid cells are then divided into five tetrahedrons each. In a post-processing step, the tetrahedrons are smoothed to align with the given surface mesh. Such a smoothed tetrahedral mesh is shown in Figure 4.2d.

The crucial point in this meshing approach is the computation of the signs for the distance field. A straight-forward approach casts rays through the object and checks, how often the ray intersects the surface mesh. Counting the intersections indicates changes from outside to inside and vice versa. This method, however, only gives accurate results for water-tight object models. For real data, which might be incomplete, and contain holes and cracks, this method does not lead to reliable results. This problem is addressed, among others, in the work of Nooruddin and Turk (2003): they also employ a ray casting and counting scheme, but consider rays from different directions, the majority vote then decides on the sign of a voxel. The approach of Spillmann et al. (2006) builds on top of that method, instead of counting ray intersections to detect transitions from inside the object to the outside, however, they consider the probability of such a transition along a ray for each voxel. This probability depends on the distance of the voxel center to the next surface feature. Thus, the sign computation is more robust for incomplete data, which proves beneficial for the meshes we obtain with an RGB-D camera. Furthermore, this approach is robust with respect to the connectivity of the meshes, for instance, meshes consisting of interpenetrating object parts as well as arbitrary unconnected triangle soups can be processed and lead to decent volumetric meshes that represent the volume occupied by an object.

The outcome of this meshing approach is a complete geometric and volumetric object representation – a surface mesh and a corresponding tetrahedral mesh – with a known transformation relative to the robot. It can be used to determine the contact point for deformation and to perform subsequent deformation simulations in the parameter estimation procedure. In the simulation framework, we perform all deformation computations on the tetrahedral mesh.

The surface mesh is coupled to the tetrahedral mesh using geometric constraints (Müller and Gross, 2004). This coupling of the surface mesh to the tetrahedral mesh guarantees that the surface mesh is also deformed. This allows us to compare it to the scanned surface mesh of the real-world object.

4.1.3 Deformation of Objects

We designed an experimental setup to measure the deformation behavior of an object with our robot Zora. For that purpose, we place the object on a table in front of the robot. The robot probes the object by moving its end effector downward, in the direction perpendicular to the table surface. This setup guarantees that the object is caught between the table and the robot, therefore the robot only deforms it and the measured forces correspond to deformations only, not to translations of the object.

The robot deforms the object with a thin wooden stick instead of its gripper. This has several reasons: first of all, the RGB-D camera requires a distance of at least 50 cm to compute depth measurements from the structured light pattern. Second, increasing the distance to the region of interest also increases the field of view and thus the part of the object that can be observed. Third, and most importantly, in this way we ensure a small point-like contact region and thereby minimize the amount of occlusion in the region of interest, the deformed surface region, due to body parts of the robot. The experimental setup and some exemplary observations are depicted in Figure 4.3. The probing procedure is as follows:

- The end effector approaches the contact point c on the object and takes a reference measurement.
- Subsequently, it moves forward in discrete steps of 1 cm, pauses shortly, and records a new measurement.
- This is done until either a maximal force of 30 N is exceeded or the robot has moved for more than 10 cm.
- Finally, the robot moves back to the initial position.

The contact point on the surface of the object can be chosen arbitrarily, we provide the robot with a set of configurations for possible interactions. For the parameter estimation procedure, explained in the next section, we need the position of the contact point on the object model. This can be computed from the known transformation of the object model

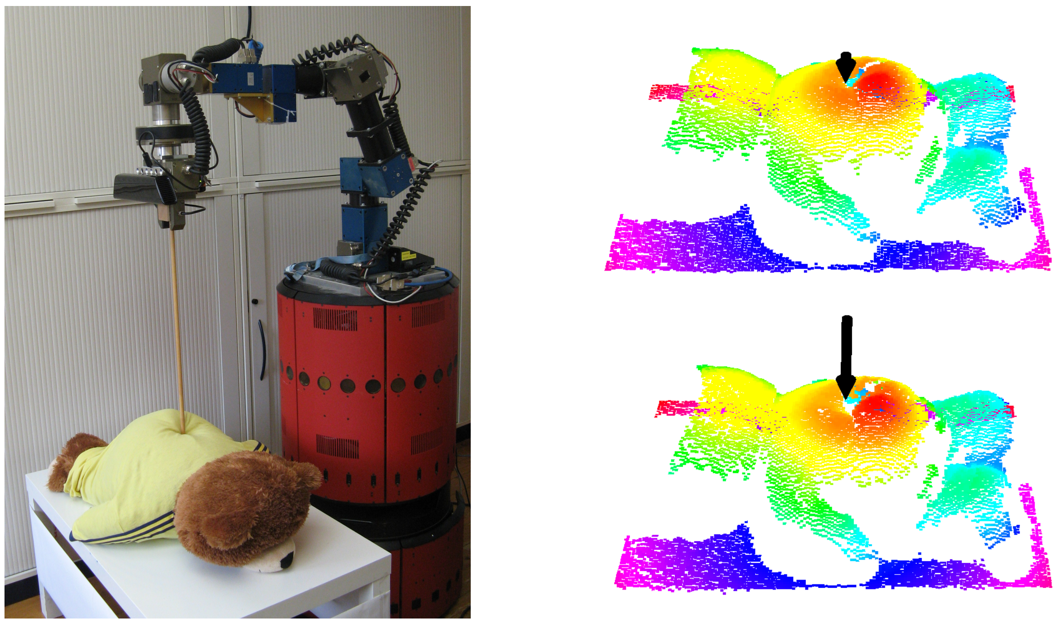


Figure 4.3: Deformation of an object: experimental setup (left) and two example measurements (right). The surface points are colored according to their depth and the magnitudes of the measured forces are indicated by the arrows.

relative to the robot and from the robot position estimate. The force acting on the object is computed by subtracting the reference force.

In each step t, we obtain a measurement $\mathbf{z}_t = (P_t, \mathbf{c}_t, \mathbf{f}_t)$, which consists of the point cloud of the deformed object surface $P_t = \{\mathbf{p}_t \mid \mathbf{p}_t \in \mathbb{R}^3\}$, the force vector $\mathbf{f}_t \in \mathbb{R}^3$ acting on the object and the contact point $\mathbf{c}_t \in \mathbb{R}^3$ on the object. In this way, we obtain a set of measurements $\{\mathbf{z}_t\}$ for a contact point. Our parameter estimation procedure, explained in the next section, only requires one observation $\mathbf{z}_t$ at a time, but collecting a set of observations allows for multiple runs and therefore a more robust estimation of the parameters.

4.2 Parameter Estimation

We have adopted a linear finite element model based on elasticity theory for deformable objects. This model allows us to predict how an object deforms when external loads are applied to it, given we know the model parameters, which are the Young modulus and the Poisson ratio for isotropic elastic materials. Conversely, the observation of a deformation and the measurement of the applied force allow us to draw conclusions on the model parameters. This can be interpreted as the solution to an inverse problem, in which the aim is to identify the model parameters from a set of measurements that best explain the system's behavior (Tarantola, 2005). More specifically, with the measurements acquired by our robot, we formulate the estimation of an object's elasticity parameters, Young's modulus E and Poisson's ratio ν, as an optimization problem in parameter space (E, ν) with an objective function that minimizes the difference between the observation and the model prediction. In a similar way, Schnur and Zabaras (1992), Kauer et al. (2002), and Becker and Teschner (2007), among others, approached this problem and determined the modeled material parameters using optimization techniques.

To specify the parameter estimation problem, we recall the governing equation solved by the FEM approximation from Section 3.2.2, which relates the applied forces and resulting displacements given in Eq. (3.29) as

$$\mathbf{F}^{\mathrm{ext}} = K(E, \nu)\mathbf{Q}, \tag{4.2}$$

with the applied force $\mathbf{F}^{\mathrm{ext}}$, displacements $\mathbf{Q}$, and the stiffness matrix $K(E, \nu)$, which depends on E and ν. The inverse problem we intend to solve can be stated as determining the stiffness matrix $K(E, \nu)$ that explains the relation between measured force $\mathbf{F}^{\mathrm{ext}}_{\mathrm{meas}}$ and measured displacement $\mathbf{Q}_{\mathrm{meas}}$

$$\min_{(E,\nu)} \|K(E, \nu)\mathbf{Q}_{\mathrm{meas}} - \mathbf{F}^{\mathrm{ext}}_{\mathrm{meas}}\|_2^2. \tag{4.3}$$

However, as the robot only observes the displacements on the boundary of the object, we cannot directly set up this equation and solve for (E, ν). Instead, we indirectly relate the observed displacements with simulated displacements by running a forward FEM simulation for a given stiffness matrix. Then, we can compare the displacements resulting from the simulation to the actually observed displacements and minimize their difference:

$$\min_{(E,\nu)} \|\mathbf{Q}_{\mathrm{meas}} - \mathbf{Q}_{\mathrm{sim(E,\nu)}}\|_2^2. \tag{4.4}$$

We use a gradient-based method to adapt the material parameters of an object and to minimize the error. In the following, we define the boundary conditions of the FEM simulation that provides us with the simulated displacements, furthermore, we specify the error function that is to be minimized.

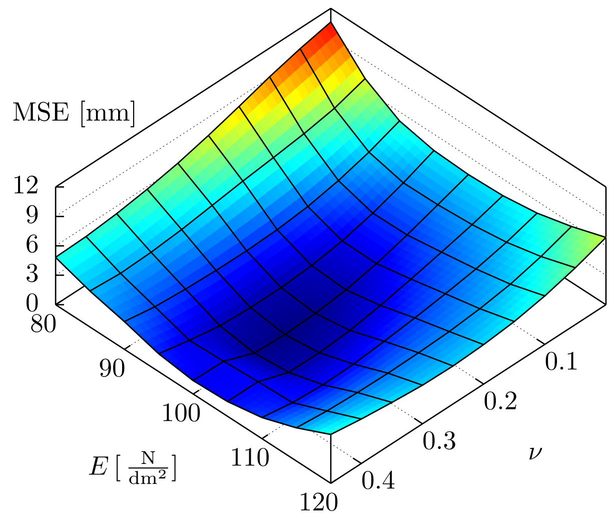
4.2.1 FEM Simulation

Before we can compute the error between observed and simulated deformation, we have to define the setting for the FEM simulation. We initialize the simulation with the tetrahedral model $\mathcal{M}$ and the corresponding surface mesh $\mathcal{P}$ of an object from Section 4.1.2. Additionally, we initialize the material parameters E, ν, from which the stiffness matrices $K(E, \nu)$ of the model elements are computed. Furthermore, we introduce boundary conditions for the simulation by fixing the nodes on the bottom side of the model, which correspond to the part of the object that is in contact with the table. Thus, the object is not moved in simulation. To start the simulation and deform the model, we apply the measured force $\mathbf{f}_t$ to the contact point $\mathbf{c}_t$ on the model, more precisely, we apply a point load $\mathbf{f}_t$ on the mass point on the tetrahedral mesh closest to the contact point. These two quantities, contact point and applied force, result from the probing experiment described in Section 4.1.3. Then, we define $\texttt{FEMSim}(\mathcal{M}, \mathbf{c}_t, \mathbf{f}_t, E, \nu)$ as a simulation run over a small amount of time steps until an equilibrium state is reached, which results in the deformed model $\mathcal{M}_{E,\nu}$ and deformed surface points $\mathcal{P}_{E,\nu}$. The deformation for a given force and contact point is governed by the material parameters E, ν of the object.

4.2.2 Error Function

Since the robot only observes the deformation of an object on its boundary, the error function for our parameter estimation procedure reflects the difference between the surface of the real deformed object and the surface deformed in simulation. Before we compute the difference between the deformed model point cloud and the observed point cloud, we align the deformed surfaces with a registration procedure. This eliminates the effects of small rotations and translations not leading to object deformations and inaccuracies in the global position estimation of the model with respect to the robot. Similar to the registration procedure described in Section 4.1.2, we register the point clouds with the Trimmed ICP algorithm that takes into account only the 80 % best point correspondences. This allows us to compute a reasonable registration, even if the deformations of model and observation do not fit together and no correct point correspondences can be found in the region of deformation.

After applying ICP, we can determine the error between the deformed model point cloud $P_{E,\nu}$ and the measured surface P_t as the mean squared error between the point correspondences



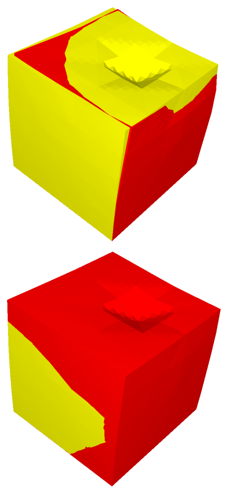


Figure 4.4: The error function in parameter space (E, ν) that is minimized in our estimation procedure for a synthetic example, in which a foam cube with "true" parameters $E = 100\,\frac{\text{N}}{\text{dm}^2}$ and $\nu = 0.3$ was deformed.

of the surfaces:

$$\text{Err}(P_{E,\nu}, P_t) = \frac{1}{|P_t|} \sum_{i \in P_t} \min_{j \in P_{E,\nu}} \|i - j\|^2, \tag{4.5}$$

where i and j refer to the corresponding points from the observed and the simulated surface, respectively. In the error function, we consider all point correspondences for the measured point cloud, in contrast to the error function minimized in the ICP algorithm. Otherwise, we would possibly ignore just the region of interest, in which the object is deformed, due to large point-to-point distances and the error function would not be particularly informative. In Figure 4.4, we illustrate the error function for a synthetic example; a cube with known parameters was deformed in simulation and the error function evaluated for a uniform sampling of parameters (E, ν). We observe, that the error function is indeed convex and contains one global minimum.

4.2.3 Parameter Optimization

With the above definition of the error function, we can apply a gradient-based method to search for Young's modulus E and Poisson's ratio ν of an object that minimize the error. We start with a random initialization of the parameters (E_0, ν_0) and iteratively adapt them based

Algorithm 1 Iterative parameter optimization

```
Input: Model $M$, force $\mathbf{f}_t$, contact point $\mathbf{c}_t$, observed deformation $P_t$
Output: Optimized parameters $E_{opt}, \nu_{opt}$
 1: Initialize $E_0, \nu_0, \Delta_0^E, \Delta_0^\nu, \mathrm{Err}_0, i = 0$
 2: loop
 3:     $E_{i+1} = E_i - \mathrm{sign}\left(\frac{\partial \mathrm{Err}_i}{\partial E_i}\right) \Delta_i^E$
 4:     $\nu_{i+1} = \nu_i - \mathrm{sign}\left(\frac{\partial \mathrm{Err}_i}{\partial \nu_i}\right) \Delta_i^\nu$
 5:     $P_{E_{i+1},\nu_{i+1}} = \mathtt{FEMSim}(M, \mathbf{c}_t, \mathbf{f}_t, E_{i+1}, \nu_{i+1})$
 6:     $\mathrm{Err}_{i+1} = \mathrm{dist}(P_{E_{i+1},\nu_{i+1}}, P_t)$
 7:     if $(\mathrm{Err}_{i+1} - \mathrm{Err}_i) < \epsilon$ or $(E_{i+1} - E_i) < \epsilon_E$ or $(\nu_{i+1} - \nu_i) < \epsilon_\nu$ or $i > \mathrm{maxIt}$ then
 8:         return $(E_{i+1}, \nu_{i+1})$
 9:     end if
10:     i++
11: end loop
```

on the direction of the gradient of the error function. Since our error function involves the simulation approach explained above, the gradient cannot be computed directly. Therefore, we approximate this term numerically: we carry out a sequence of deformation simulations by applying the measured force to the model and by varying E and ν locally.

We adapt the parameters based on the Resilient backpropagation (Rprop) update rule that was introduced by Riedmiller and Braun (1993) in the context of learning weights for neural networks. In this update rule, a step size Δ_k for each parameter k is adjusted individually in each iteration step based only on the direction, not on the magnitude of the gradient. More precisely, the step size for each parameter is increased in each iteration i by a factor $\eta^+ > 1$ if the gradient direction does not change, that is if a minimum of the error function is approached, and it is decreased by $\eta^- < 1$ otherwise, that is if a minimum of the error function is overstepped:

$$\Delta_E^{(i)} = \begin{cases} \eta^+ \Delta_E^{(i-1)}, & \text{if } \frac{\partial \mathrm{Err}^{(i)}}{\partial E} \frac{\mathrm{Err}^{(i-1)}}{\partial E} > 0 \\ \eta^- \Delta_E^{(i-1)}, & \text{if } \frac{\partial \mathrm{Err}^{(i)}}{\partial E} \frac{\mathrm{Err}^{(i-1)}}{\partial E} < 0 \\ \Delta_E^{(i-1)}, & \text{else} \end{cases} \tag{4.6}$$

The weight factors are set to $\eta^+ = 1.2$ and $\eta^- = 0.5$, as these values have been identified experimentally to lead to a robust convergence behavior for different types of problems (Riedmiller and Braun, 1993). This procedure is robust with respect to the initialization of the step size, as the step size quickly adapts to the problem at hand. Furthermore, it is robust to numerical inaccuracies, as only the direction, not the magnitude of the gradient is considered. Thus, it allows for a fast convergence of our estimation procedure. Algorithm 1 summarizes the main steps of the iteration routine. We consider the estimation procedure converged if either the error improvement is below a given threshold ϵ, or if the parameter adaptations are below given accuracy thresholds ϵ_E and ϵ_ν for both parameters E and ν in subsequent iteration steps.

4.3 Limitations of the Deformation Model

Our deformation model describes linearly elastic, isotropic, and homogeneous material. Most real materials, however, show some nonlinear, anisotropic, inhomogeneous, and also viscoelastic behavior as observed by Bickel et al. (2009); Kauer (2001); Lang (2001), among others. To investigate these effects in our setting, we carried out a deformation experiment with a foam cube. The cube was compressed between two plates, one of which was moved by the robot manipulator in 1 cm increments. After each movement of the manipulator, the deformation was kept constant for some seconds. The curve in Figure 4.5 shows the recorded force measurements over time. We observe that the force decreases over time when the deformation is held constant. Furthermore, we can observe a hysteresis, as the measured force is significantly larger when loading the object compared to unloading the object for the same position of the manipulator, and the same deformation, respectively. The hysteresis is illustrated in a force-displacement plot of this experiment in Figure 4.6. These effects can be explained by viscoelastic behavior of the object: some of the deformation energy is dissipated in the object, and the material partially adapts to the deformation. Such effects are not accounted for in our deformation model. We also observe that the force-displacement curve is not perfectly linear.

In our parameter estimation experiments, we only use the force measurements obtained when loading the object, that is when the manipulator moves forward to deform the object. Since these forces are larger, we get a more conservative estimate of the force required to deform an object when using the learned models for robot motion planning. We do not consider force measurements obtained when the robot moves back. Furthermore, we determine an average of the force measurements over a time frame of two seconds, when we record a force-deformation sample. In our experiments, we evaluate how accurately we can determine the material parameters of real objects. Furthermore, we evaluate how robust these

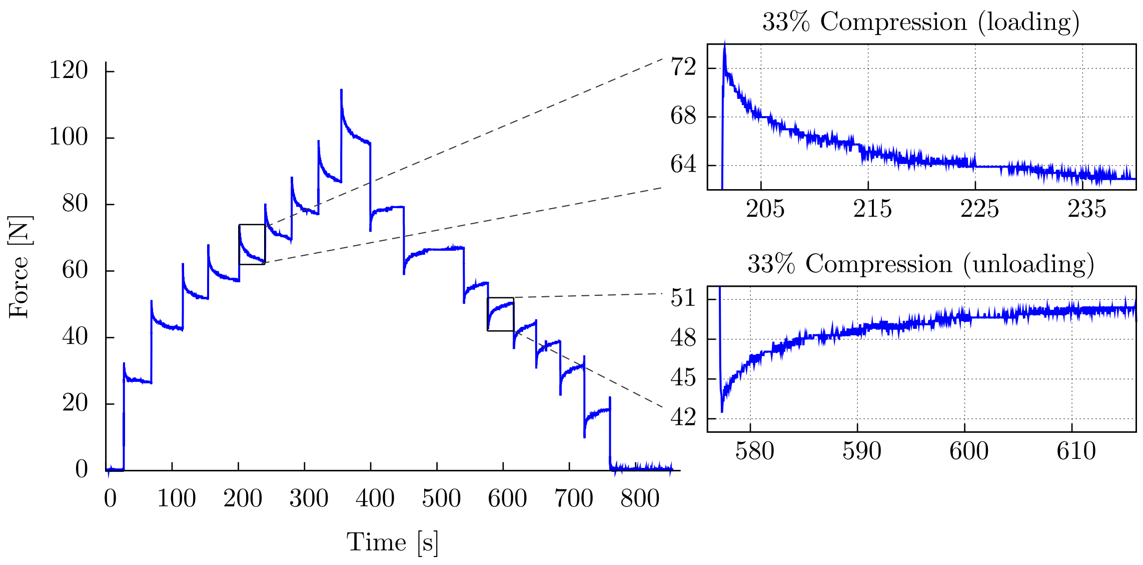


Figure 4.5: Force observations for a compression experiment with a foam cube. We notice that the measured force is time-dependent for a constant deformation, which can be explained by some viscoelastic behavior of the material.

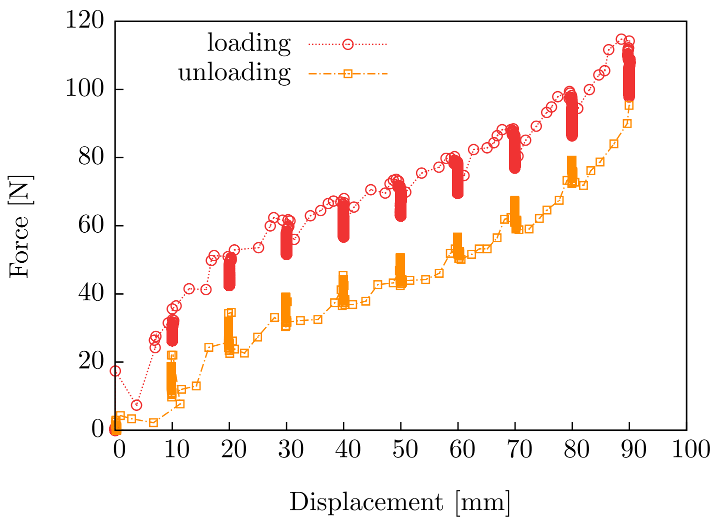


Figure 4.6: The force-displacement curve for the compressed foam cube (Figure 4.5) shows a hysteresis.

estimations are, with respect to predicting different deformations and forces. Heterogeneous material properties could in principle be accounted for in our model, by determining the material parameters for different heterogeneous parts of the object individually and associating them with the corresponding tetrahedral elements of the model. We have not realized such a model yet, but we evaluate the error that is introduced by assuming homogeneity.

4.4 Experimental Results

We carried out different experiments to evaluate our parameter estimation procedure with observations of object deformations obtained from simulations and from interactions with real objects. In simulation experiments, we evaluated the accuracy and precision of the parameter estimation procedure under the influence of different sources of noise. For the observed deformations of real objects, we evaluated the robustness of the parameter estimation as well as the error in predicting new force measurements for the estimated parameters.

4.4.1 Simulation Experiments

We evaluated our parameter estimation procedure under controlled conditions in a simulation experiment. Our test object is a cube with an edge length of 20 cm and true material parameters $E = 100\,\frac{\mathrm{N}}{\mathrm{dm}^2}$ and $\nu = 0.3$. The model consists of 625 tetrahedrons and the surface mesh consists of 2,646 points. We deformed the cube in simulation and generated a test data set consisting of 10 force-deformation samples with linearly increasing force in the range of 3 to 30 Newton. In different runs, we evaluated the results of the estimation procedure under the influence of different noise characteristics. We identified three different sources of noise:

(1) Noise in the RGB-D measurements, which is around 2 mm for distances below 1 m, we assume $\sigma_p \sim 2.5\,\mathrm{mm}$.

(2) Noise in the force measurements, which contains a force-dependent noise component of $\sim 5\,\%$ as specified by the manufacturer and a white noise component with a magnitude of approximately 1 N, as can be observed, for instance, in Figure 4.5: $\sigma_f \sim 0.05|\mathbf{f}| + 1\,\mathrm{N}$.

(3) Noise in the estimation of the contact point: $\sigma_c \sim 20\,\mathrm{mm}$.

In each run, we evaluated the iterative parameter estimation procedure for all 10 force-displacement samples. Run 1 to 3 consider the three types of noise mentioned above individually and run 4 considers a combination of all types of noise. Figure 4.7 summarizes the results in terms of the error in the estimated Young's modulus, Poisson's ratio and the residual mean square error (MSE) after convergence of the estimation. Furthermore, it illustrates the evolution of the parameters and the error in one learning run for the different

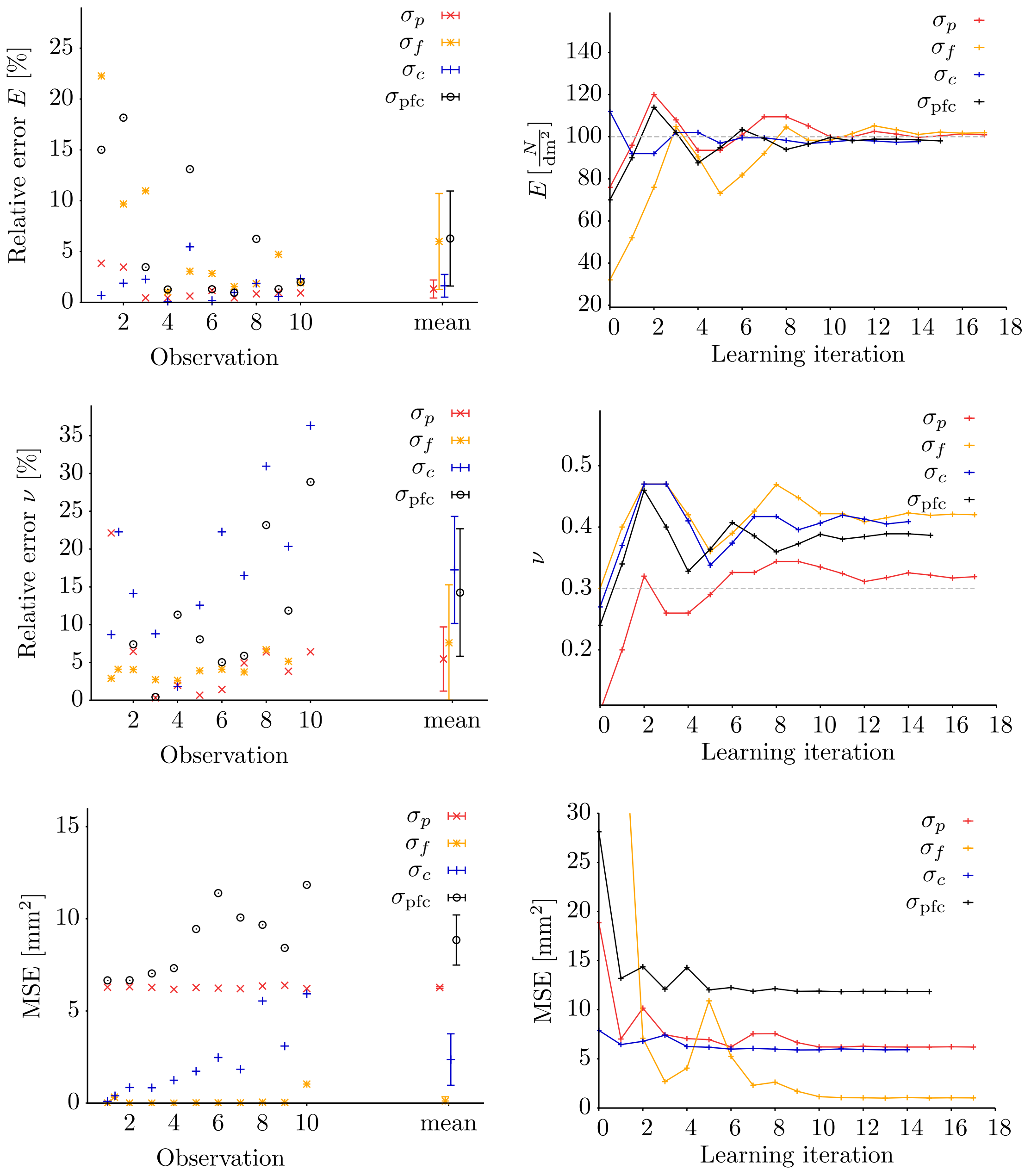


Figure 4.7: Parameter estimation results for simulated data sets consisting of different force-deformation observations corrupted with different sources of noise. The plots in the left column show the relative errors in the estimated Young's modulus E (top) and Poisson's ratio ν (middle), furthermore the residual MSE of the surface meshes after convergence of the estimation procedure (bottom). The plots in the right column illustrate the evolution of the corresponding quantities in learning runs for sample 9, for deformations with $\mathbf{f} \approx 27\,\text{N}$.

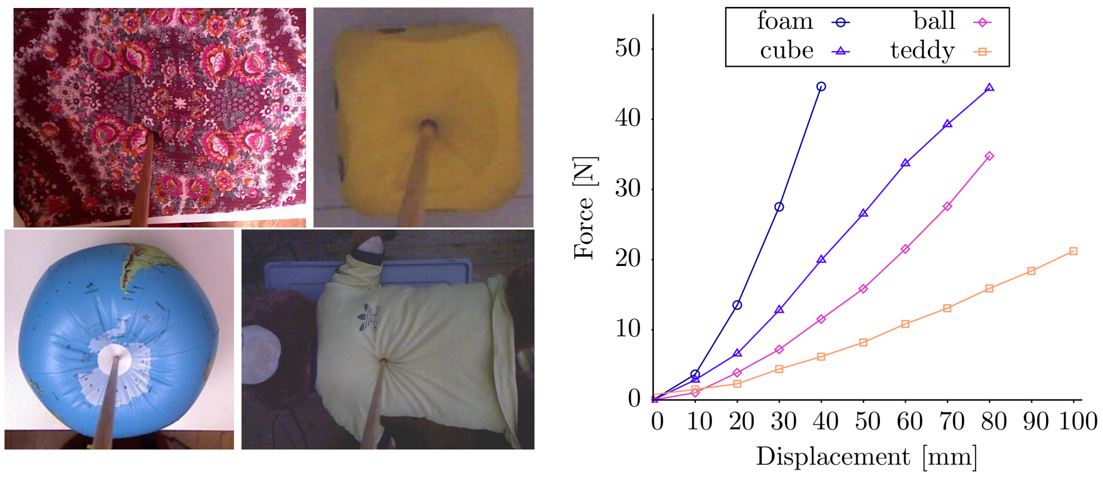


Figure 4.8: Real objects, for which we estimated the material parameters: a foam mat, a foam cube, an inflatable ball, and a plush teddy. The foam mat and the plush teddy bear are covered with cloth to make them observable for the RGB-D camera. The plot on the right shows the force-displacement curves for the recorded measurements of each object.

noise settings. From the results, we can make some interesting observations. The observation noise (run 1) does not seem to affect the parameter estimation, the parameters are estimated accurately for all samples while the residual error after convergence corresponds to the observation noise. Noise in the force measurements (run 2) naturally leads to a larger error in the estimated parameters. This error is more pronounced for samples with smaller deformations and forces due to the white noise component in the force observation. The residual error, in contrast, is very small, the estimated parameters simply express a different force-displacement relation. An error in the contact point leads to a different deformation of the model, hence, the observation can never be entirely consistent with the deformed model. This error is more pronounced for larger deformations. For a combination of all errors in run 4, the relative error in the estimation of the Young modulus is still below 10 %, while for Poisson's ratio it is around 15 %. Thus, our estimation procedure allows to identify the material parameters from force-deformation observations.

4.4.2 Parameter Estimation for Real Objects

We evaluated our parameter estimation approach on observations of four different real objects: a foam mat with a size of 50 x 80 x 5 cm, a foam cube of edge length 15 cm, an in-

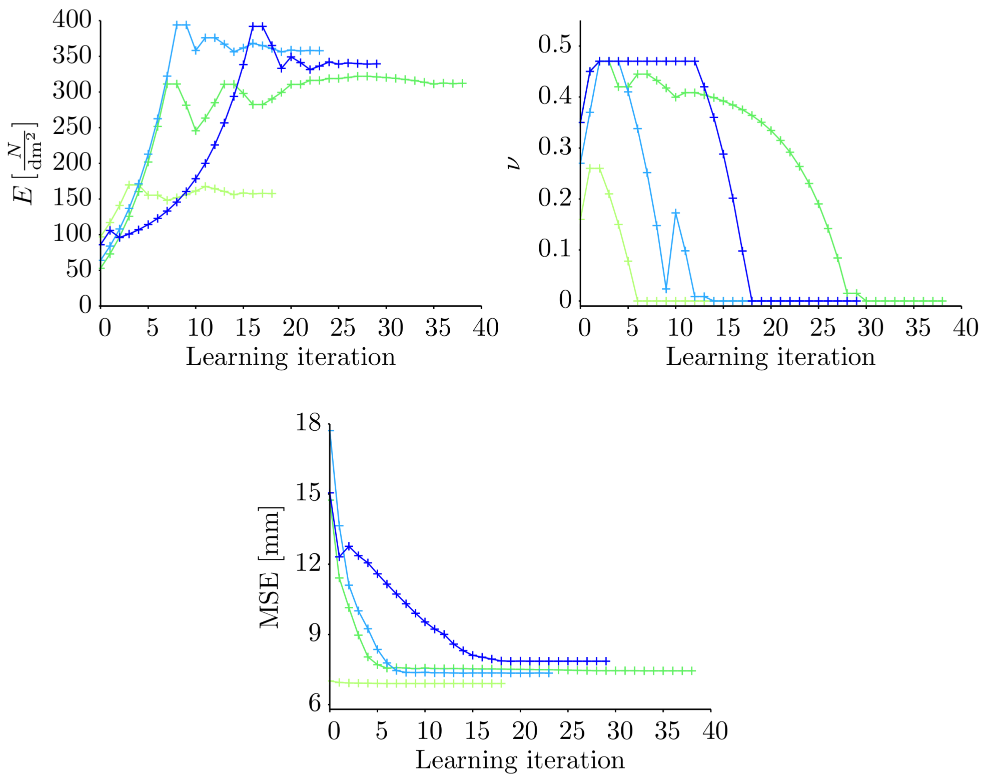
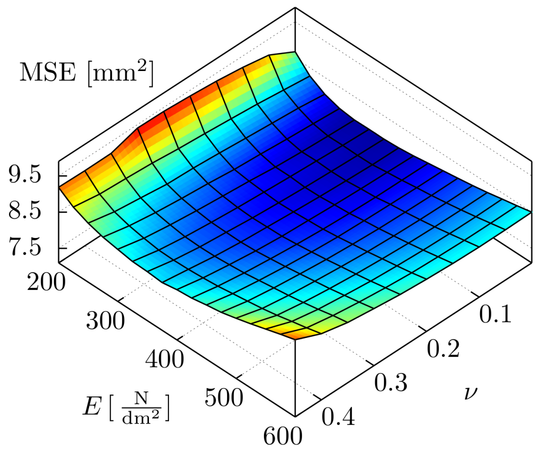


Figure 4.9: Parameter estimation for the foam mat: The plots show the adaptation of the parameters over the learning iterations (top row) and the MSE of the registered surface meshes (bottom). The colors encode the applied force (green: low force, blue: high force).

flatable ball with a diameter of approximately 40 cm, and a plush teddy bear with a height of approximately 50 cm. For each object, we recorded a test series of force-deformation samples with increasing force for one contact point. For the teddy bear, we additionally considered several contact points. The objects and the corresponding force-displacement curves for the recorded samples, with the displacement derived from the manipulator motion, are shown in Figure 4.8. In the following, we present parameter estimation results for each object.

Foam mat

We recorded a series of four force-deformation samples for one contact point on the foam mat, and estimated the material parameters for each of the samples individually. The evolution of the parameters and the error in the individual learning runs are shown in Figure 4.9.



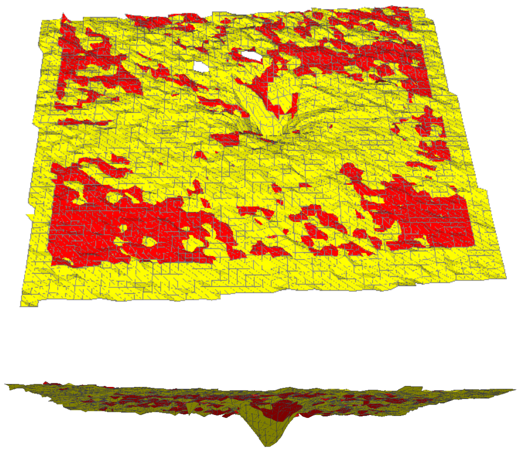


Figure 4.10: The error function for foam sample 4 ($\mathbf{f} = 44\,\mathrm{N}$) for a uniform sampling of parameters (E, ν): the error varies less with changing ν than with changing E. The global minimum at $E = 300\,\frac{\mathrm{N}}{\mathrm{dm}^2}, \nu = 0$ agrees with our gradient-based estimation ($E = 339.5\,\frac{\mathrm{N}}{\mathrm{dm}^2}, \nu = 0$). The corresponding registered surface meshes are shown on the right.

While the estimated values for the Young modulus correspond well for the last three samples, the estimation for the first measurement converges to a considerably smaller value. This can be explained one the one side with the nonlinearity in the force-displacement curve, and on the other side with very small deformation region that is hardly noticeable in the error function – it almost gets lost in the measurement noise. If we discard sample 1 as outlier, and average over the remaining three samples, we obtain an estimate of $340.2\,\frac{\mathrm{N}}{\mathrm{dm}^2} \pm 88.2\,\frac{\mathrm{N}}{\mathrm{dm}^2}$ for Young's modulus, if we consider 95 % confidence intervals.

The estimation for Poisson's ratio converges to zero for each sample. This is somewhat surprising, since the Poisson ratio of foam is reported to be in range of 0.1 to 0.3 in general. To gain more insight into this behavior of our estimation procedure, we consider the error function for sample 3 in Figure 4.10. This error function was obtained for a uniform sampling in the parameter space and is mainly influenced by the value of the Young modulus, while a change in the Poisson ratio leads to comparably small changes in the error function. This could be explained by the fact that the deformation is observed from above, and since the foam mat is larger than the field of view of the camera, a possible extension of the object transverse to the applied force unfortunately cannot be observed with our sensor setup.

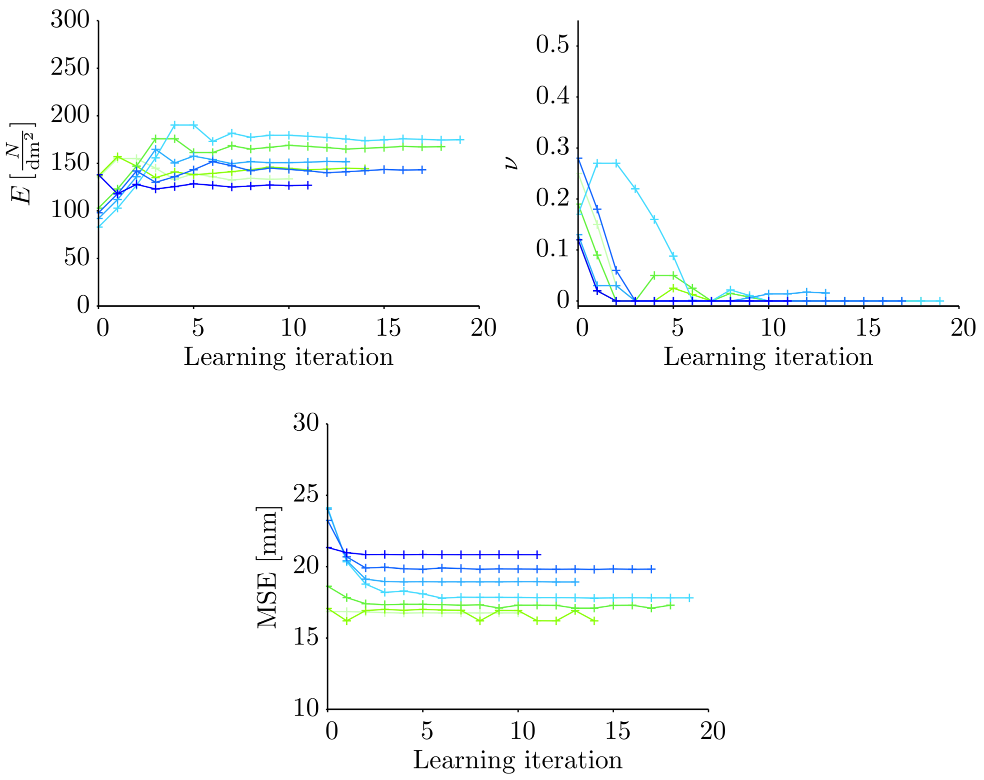


Figure 4.11: Parameter estimation for the foam cube: The plots show the adaptation of the parameters over the learning iterations (top row) and the MSE of the registered surface meshes (bottom). The colors encode the applied force (green: low force, blue: high force).

Foam cube

In addition to the foam mat, we examined a toy cube consisting of a different type of foam. It is softer, as can already be observed from the force-displacement curve (Figure 4.8). The learning curves for the seven samples are shown in Figure 4.11. The Young's modulus estimated for different applied forces varies and is in the range of $148.9\,\frac{\mathrm{N}}{\mathrm{dm}^2} \pm 17.2\,\frac{\mathrm{N}}{\mathrm{dm}^2}$. Similar to the foam mat, Poisson's ratio converges to zero, as an extension of the object perpendicular to the camera is hardly observable. The residual error of the registered surface meshes increases significantly with increasing force. Figure 4.12 illustrates the deformation model determined for the foam cube with the last observation of the data set. It shows the deformed tetrahedral model for the determined material parameters and the registered surface meshes, furthermore it illustrates the point-wise error for the observed deformation. We can see that the error is smaller in the deformed region than in the border regions with the edges of the cube, thus the deformation model fits the observation quite well.

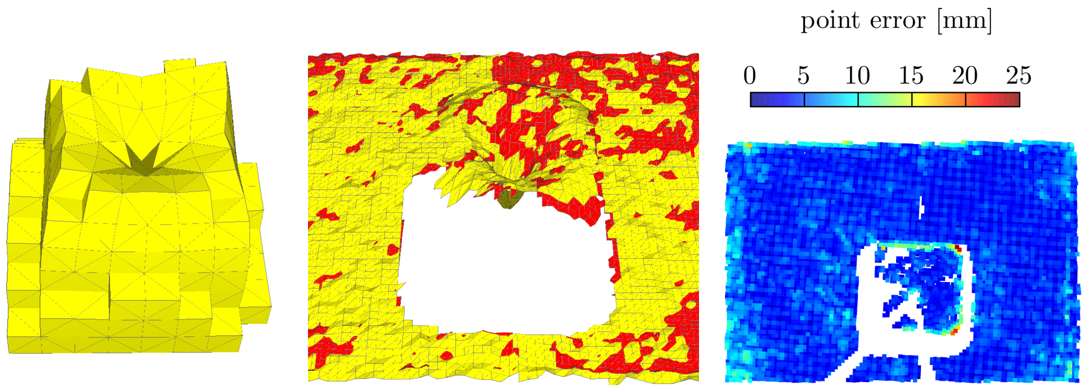


Figure 4.12: The deformation model for the foam cube estimated with sample 7, with a deformation force of $\mathbf{f} = 44\,\mathrm{N}$: The tetrahedral mesh deformed with the estimated parameters $E = 126.9\,\frac{\mathrm{N}}{\mathrm{dm}^2}, \nu = 0$ is shown on the left, the registered surface meshes, the observed surface mesh (red) and the deformed model (yellow) are shown in the middle. The plot to the right illustrates the point-wise error for the observed surface mesh.

Inflatable ball

The inflatable ball has a larger diameter and a smaller force is required to deform it. Thus, we were able to acquire eight force-deformation samples in total. The material parameters were estimated for each sample individually. Figure 4.13 illustrates the results for the individual estimation runs. The estimated Young's modulus is in the range of $65.5\,\frac{\mathrm{N}}{\mathrm{dm}^2} \pm 8.1\,\frac{\mathrm{N}}{\mathrm{dm}^2}$ and has a low variance over the different runs. The variance in the estimated Poisson's ratio, in contrast, is rather large (0.27 ± 0.12). The residual error for the registered meshes is notably larger than for the foam mat, in particular for larger deformation forces. Figure 4.14 shows the registered surface meshes for sample 8 and its model deformed with the estimated parameters. The larger error could be explained by the fact that the model never entirely fits the observed deformation. An idea to improve the model error could be to adapt the resolution of the underlying tetrahedral model used to compute the deformation. In our experiments, however, we have not considered this possibility. We generated tetrahedral meshes with approximately 1,000 to 2,000 elements to bound the computation time of the parameter estimation.

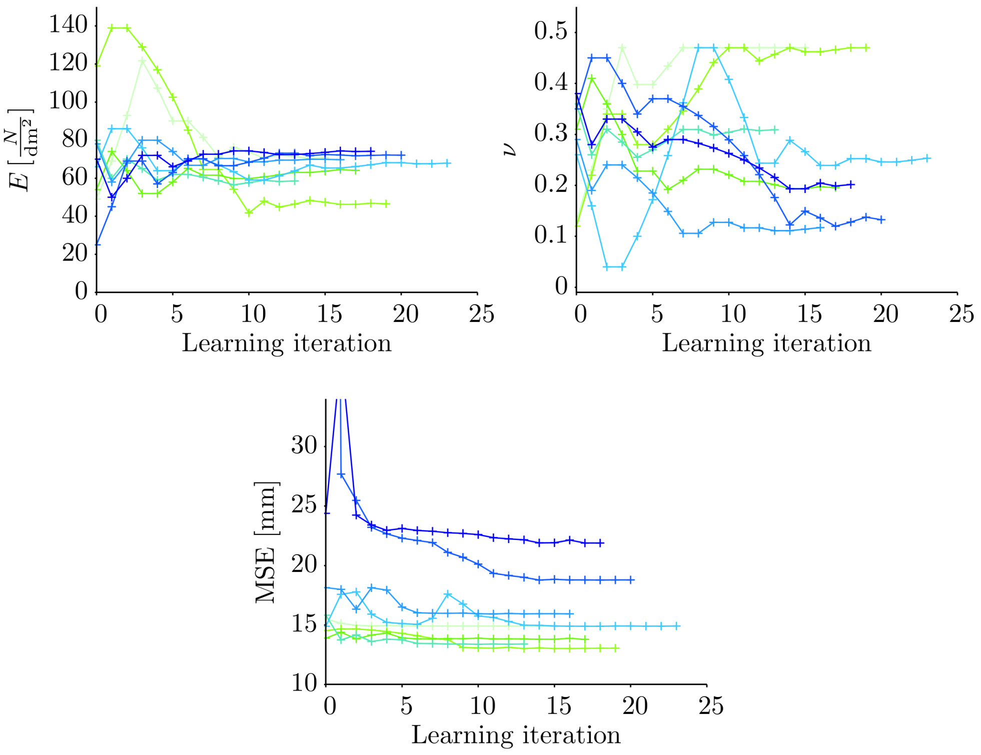


Figure 4.13: Parameter estimation for the inflatable ball: The plots show the adaptation of the parameters over the learning iterations (top row) and the MSE of the registered surface meshes (bottom). The colors encode the applied force (green: low force, blue: high force)

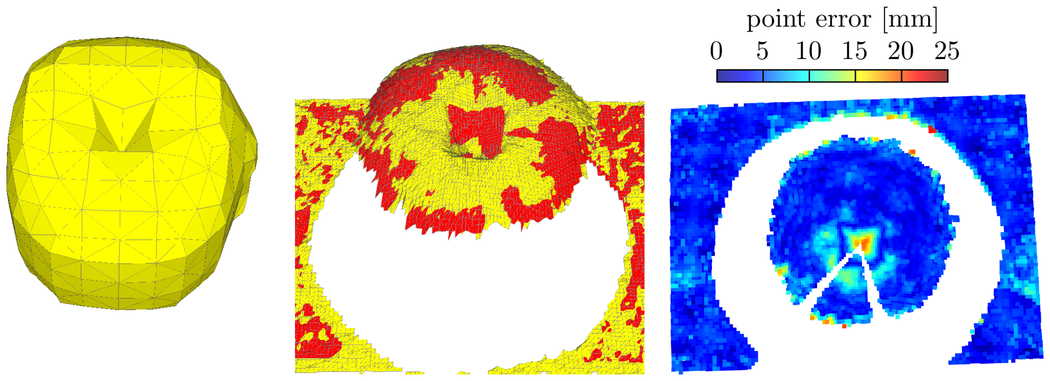


Figure 4.14: Registered surface meshes for the parameter estimation result of ball sample 8: the observed surface mesh deformed with $\mathbf{f} = 35\,\mathrm{N}$ (red) and the model with the estimated parameters (yellow, $E = 74.1\,\frac{\mathrm{N}}{\mathrm{dm}^2}$, $\nu = 0.2$) do not correspond perfectly. The tetrahedral mesh used to compute the deformation is shown on the left and the point-wise error for the observed surface mesh is shown on the right.

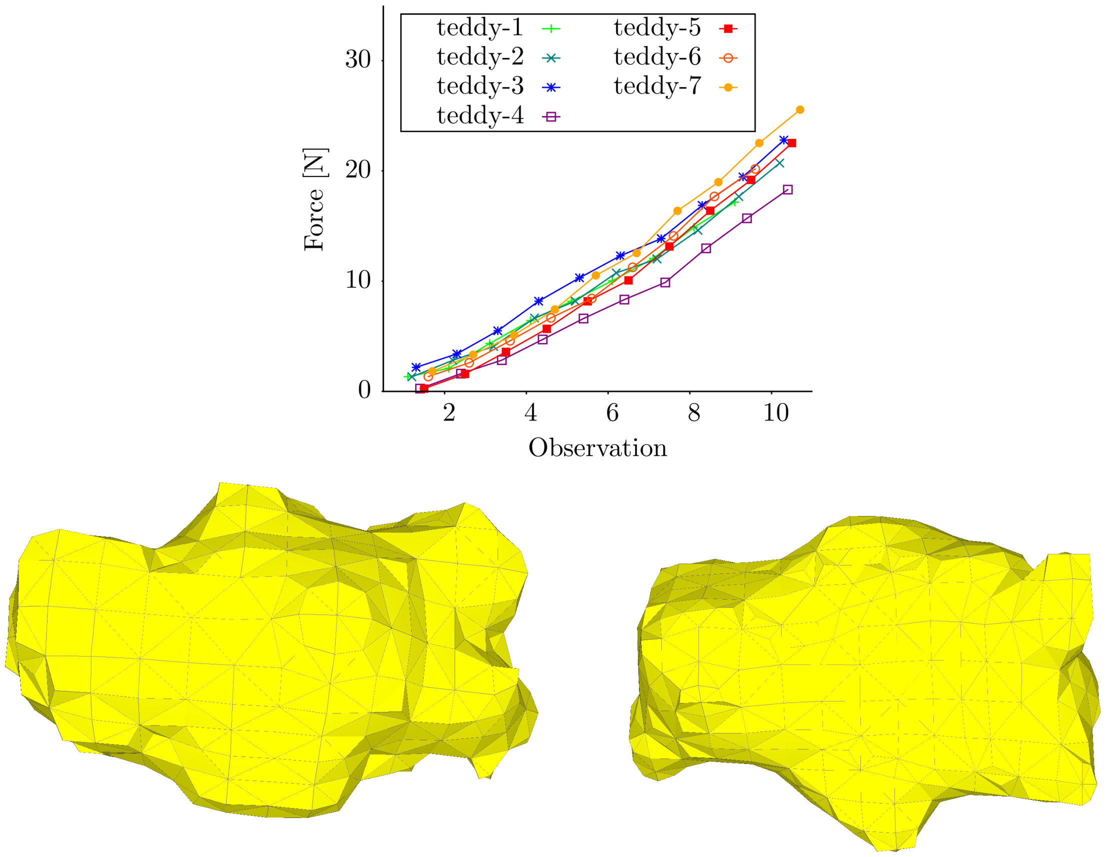


Figure 4.15: Deformation experiments with the plush teddy bear: the force-deflection curves show the recorded force observations for seven different contact points (top). The teddy was deformed at contact points on its back (1 and 2), head (3 and 4), belly (5 and 6) and chest (7). Accordingly, we generated two different models of the teddy, lying on its belly (bottom left) and on its back (bottom right).

Plush teddy

The plush teddy bear is a quite large and inhomogeneous object. To study our assumption of homogeneous material in more detail, we acquired several test series of force-deformation observations at different contact points on its back, head, belly and chest. Accordingly, we generated two different volumetric meshes for the parameter estimation procedure, one representing the teddy lying on its belly and one representing it lying on its back. These models, together with the force-deflection curves for the different deformation experiments are illustrated in Figure 4.15. We estimated the material parameters for each force-deformation observation and each contact point individually. The results are summarized in Figure 4.16. For all contact points, the variance in the estimated parameters is lower, if larger forces are

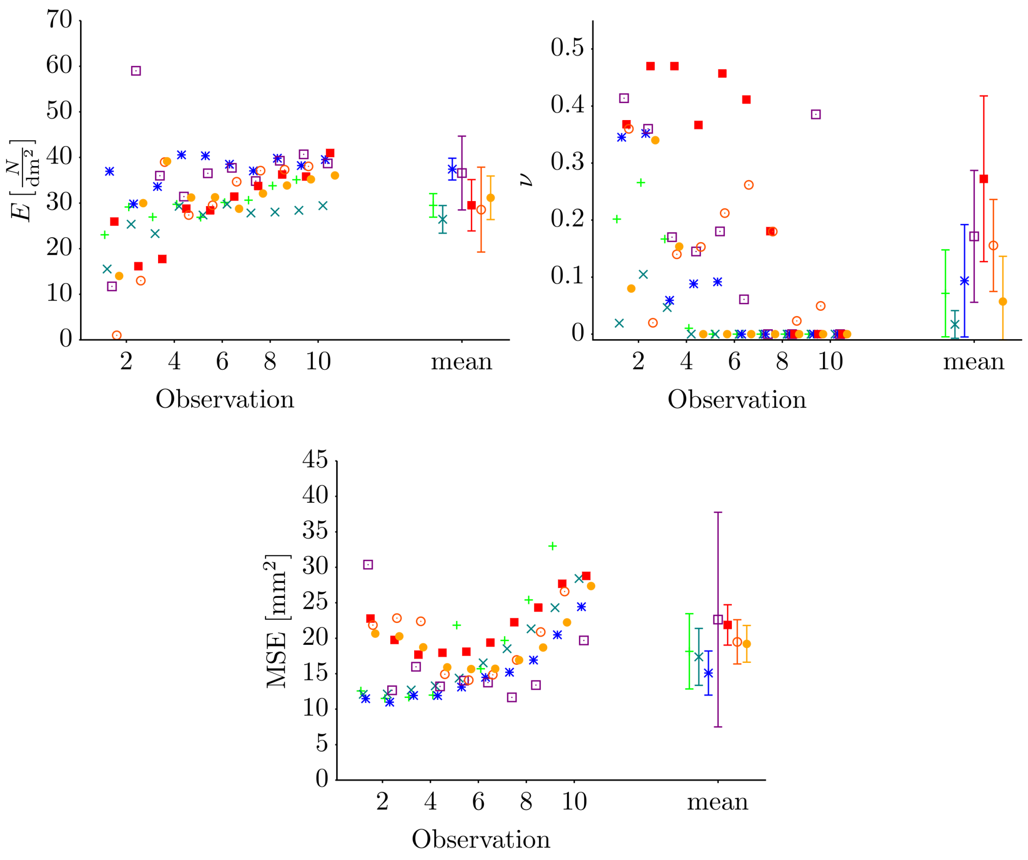


Figure 4.16: Parameter estimation results for the plush teddy bear for different force-deformation observations on different contact points. The estimated parameters, Young's modulus (top left) and Poisson's ratio (top right) and the residual MSE (bottom) are shown for the individual observations, together with the mean and confidence interval for each contact point.

applied. This is probably related to a larger deformation region in the surface observation, which can be better matched with the deformed model. Furthermore, the estimated parameters, in particular the Young modulus, vary for different contact points, the assumption of homogeneous material is obviously not applicable for this object. The residual error for the registered surface meshes tends to get larger for larger applied forces, which could be related to the mesh resolution of the underlying volumetric meshes. We illustrate the learned models and the registered observed surface meshes for different estimation runs, Figures 4.17, 4.18 and 4.19 show different estimation runs with a large, small and medium residual error, respectively. The parameters estimated in the different experiments, however, are still similar.

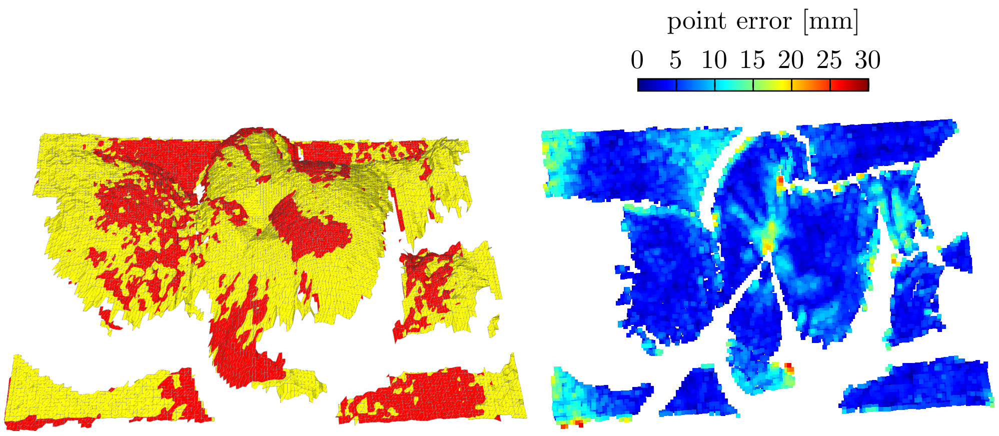


Figure 4.17: Bad model fit: teddy-1, sample 9, $E = 35.1 \frac{\text{N}}{\text{dm}^2}, \nu = 0, \text{MSE} = 33\,\text{mm}^2$.

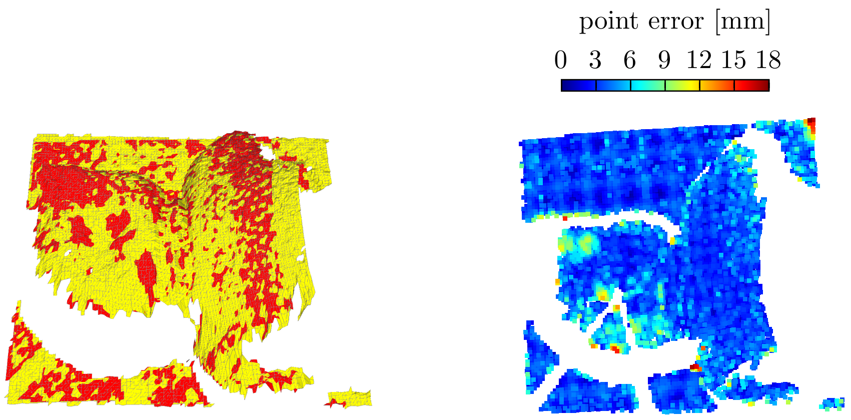


Figure 4.18: Good model fit: teddy-4, sample 8, $E = 39.3 \frac{\text{N}}{\text{dm}^2}, \nu = 0, \text{MSE} = 13.4\,\text{mm}^2$.

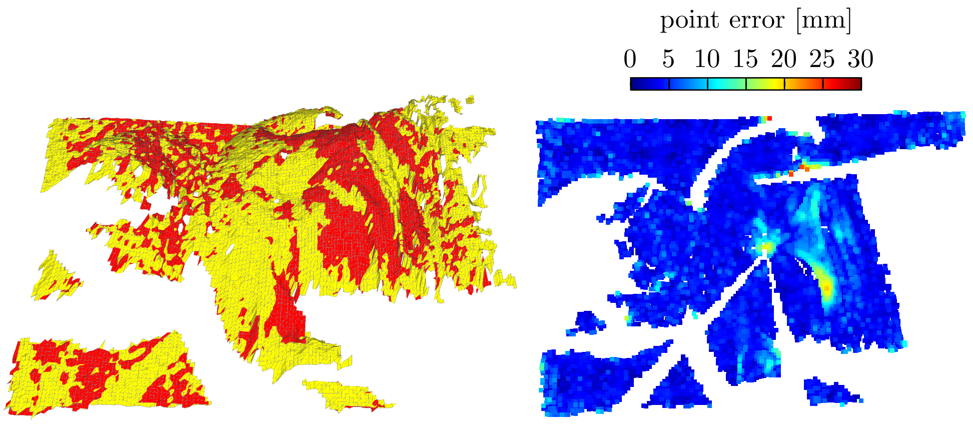


Figure 4.19: Medium model fit: teddy-5, sample 6, $E = 31.4 \frac{\text{N}}{\text{dm}^2}, \nu = 0.4, \text{MSE} = 19.4\,\text{mm}^2$.

	$\varnothing E$ ($\frac{\mathrm{N}}{\mathrm{dm}^2}$)	$\varnothing \nu$	MSE (mm)	Force error (%)
Foam (3 samples)	340.2 ± 88.2	0.0 ± 0.0	7.5 ± 0.8	10.0 ± 15.2
Cube (7 samples)	148.9 ± 17.2	0.002 ± 0.005	18.2 ± 1.7	12.9 ± 14.0
Ball (8 samples)	65.5 ± 8.1	0.27 ± 0.12	15.8 ± 2.7	12.5 ± 17.8
Teddy (9 samples)	29.5 ± 3.0	0.07 ± 0.08	18.1 ± 6.0	12.7 ± 10.5

Table 4.1: Parameter estimation results for different real objects. We determined the average over different runs with different forces applied to one contact point.

Validation of the learned models

We determined the material parameters for each object in a test series with several force-deformation samples. The means of the estimated parameters together with their 95 %-confidence intervals over the different runs already give an indication on the reliability of the estimation. They are summarized in Table 4.1 for all objects we considered in our experiments. In a validation experiment, we additionally evaluated how well the determined material parameters allow us to predict the measured forces. To this end, we performed a leave-one-out-validation for each test series. A test series recorded for one contact point consists of x force-deformation samples with increasing force. In the validation experiment, we used $(x - 1)$ samples to determine the averaged material parameters, and the remaining sample to evaluate how accurately the measured force can be predicted assuming these parameters. In detail, we determined the force that minimized the difference between the observed surface and the simulated deformation. Table 4.1 lists the averaged force prediction errors for all objects. The forces could be predicted with an error of approximately 10 to 15 %. Thus, the learned models can be useful in predicting the force a robot has to expend when deforming objects, although we neglect different material effects, such as viscoelasticity and nonlinearity.

4.5 Summary

In this chapter, we presented an approach to learn deformation models of real objects. To this end, we set up a manipulation robot to interact with deformable objects and to observe them under applied forces. With the data acquired by our robot, we showed how to establish a three-dimensional geometric model of the object and how to determine its material parameters. The material parameters are optimized using an inverse finite element approach such that they best match the observations. We evaluated our approach in extensive experiments on simulated and real-world data. We estimated the material parameters of different real objects and demonstrated that the learned models can accurately predict their deformations and forces.

The resulting models can be used in deformation simulations, for instance in virtual reality or game applications and relieve users from the need to hand-tune parameters for visually pleasing behavior. In the next chapter, we apply the learned models to describe obstacles in the environment of our robot. This allows us to carry out deformation simulations of robot trajectories that might lead to object deformations and to appropriately consider these additional costs when planning a robot's motions.

5 Deformation Cost Functions for Motion Planning

When planning robot motions, we want to consider the costs that the robot introduces by deforming obstacles in its environment. To achieve this, we carry out deformation simulations of the corresponding robot trajectories using the physical simulation framework discussed in Chapter 3. The simulation, in turn, requires appropriate deformation models of the obstacles in the robot's environment, which need to be determined beforehand – for instance, using our parameter estimation approach presented in Chapter 4. With the simulation framework and deformation models of obstacles, we define a measure for the deformation costs of a robot trajectory, which considers the deformation energy of objects.

Planning the motion to a given goal typically requires evaluation of many alternative trajectories. The simulations of the corresponding trajectories, however, are time-consuming. Thus, when given a goal point, the robot will spend some time deliberating on the best way to get there before it can actually set out for the goal. For robots navigating in real-world environments, this is not desirable, as they are expected to quickly respond to tasks. Therefore, we introduce the concept of object-dependent deformation cost functions that can be precomputed for stationary objects and significantly speed up the planning process. We model these deformation cost functions using Gaussian process (GP) regression and discuss how to evaluate trajectories generated by the planner within the GP framework.

5.1 Deformation Costs of a Robot Trajectory

To measure the costs the robot introduces by deforming an object and thereby consuming additional energy, we consider the potential elastic energy of an object, as given in Eq. (3.14). The elastic energy describes the work done, that is the force applied to deform the object. It is computed at each time step of the simulation to derive the internal forces acting inside the object and restoring its undeformed state. In an equilibrium deformation state, the internal forces are equal to the external forces acting on the object. Therefore, the potential elastic energy provides an intuitive measure for the deformation costs.

Objects are modeled using volumetric meshes consisting of tetrahedral elements. Furthermore, their elastic behavior is described by a set of material parameters, in our model introduced in Section 3.2.1, these are Young's modulus and Poisson's ratio. In the finite element method (FEM), the elastic energy of an object is computed per tetrahedral element at each time step (see Eq. (3.24)). To obtain the deformation costs for an object O consisting of elements $\{e_i\}$ in a given state, we define the inner energy Λ_O induced by the robot r in a given configuration ζ to be the sum over the inner energies of all elements e_i: $\Lambda_O(r,\zeta) := \sum_{e_i \in O} \Lambda_{e_i}(r,\zeta)$.

We define the deformation costs of a given robot configuration ζ as $C_{def}(\zeta) := \sum_{O \in \mathcal{W}} U_O(r,\zeta)$ by summing over the energies of all objects O in the workspace $\mathcal{W}$ in contact with the robot.

The deformation costs of a robot trajectory can then be computed by integrating the deformation costs over the robot movement on the path. The execution of the path is simulated in discrete time steps. Thus, the total deformation costs of a path Γ in the environment result in the sum over the deformation costs of all objects that are deformed by the robot while it is moving on the path in discrete time steps t_i, thereby assuming corresponding configurations ζ_i:

$$C_{def}(\Gamma) = \sum_{\zeta_i \in \Gamma} C_{def}(\zeta_i)\,. \tag{5.1}$$

Hence, the deformation costs of a path depend on the sequence of configurations that the robot assumes during execution of the path, furthermore on the material properties of objects in contact with the robot.

5.2 Object Deformation Cost Functions

In our definition above, we have seen that the deformation costs of a trajectory are a function of the robot configurations and the objects on the way. When the planner evaluates trajectory hypotheses, it could in principle carry out deformation simulations for each hypothesis online. In environments, in which objects can be moved by the robot and potentially new deformations resulting from interactions between obstacles need to be considered, this is the only possibility to determine an optimal trajectory. Finite element simulations, however, are time-consuming, and typically many path hypotheses need to be evaluated by the planner. Thus, in practical applications, it is not desirable to perform these simulations online.

To increase the efficiency of the planner, we consider a simplified scenario by making two preliminary assumptions:

(1) We restrict ourselves to an environment, in which objects can be deformed by the robot, but cannot be moved. This assumption is valid for objects that are partially fixed, such as curtains that are fixed to the ceiling, or plants with deformable branches or leaves in (rigid) plant pots that should not be moved by the robot.

(2) We ignore interactions between different objects in the environment, which requires that objects are placed sufficiently far away from each other.

With the assumptions stated above, the use of a roadmap-based planner, which precomputes a graph for a static environment and efficiently answers multiple path queries, is justified. In principle, it is possible to precompute the deformation costs of edges in the roadmap by performing the corresponding finite element simulations when generating the roadmap. This saves computation time when answering path queries but has the disadvantage that recomputations are necessary whenever the environment changes, for instance, when an object is moved to a new location, or the robot is deployed in a new environment containing similar objects. To overcome the shortcomings stated above, we propose a different approach: we introduce the concept of deformation cost functions for individual objects. Such object deformation cost functions are defined for robot trajectories relative to the object. They can be learned once for each type of object and are independent of the actual locations of obstacles. The availability of deformation cost functions is advantageous if there are many instances of the same object type, or if the environment changes often by circumstances outside the control of the robot. This could be the case in environments populated by humans, someone might move the flower pot to a sunnier spot.

The idea of our approach is to generate some trajectory samples relative to an object and perform the corresponding FEM simulations in a preprocessing step. Given this set of training samples, the problem of estimating the deformation costs introduced by a robot can then

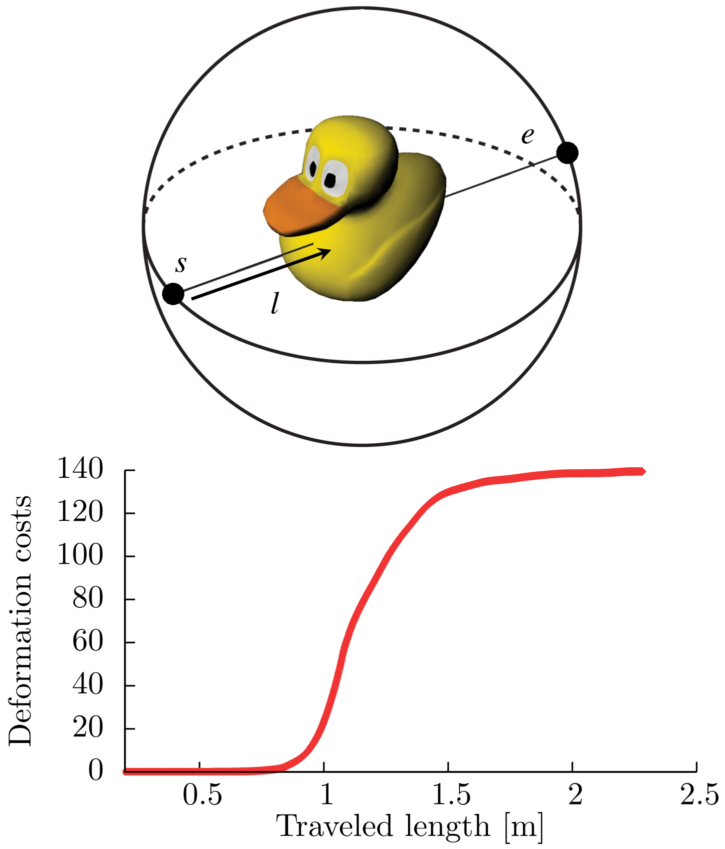


Figure 5.1: Trajectory parametrization: the linear trajectory is described by a starting point s and end point e on a virtual sphere around the deformable object. Additionally, we consider the traveled distance l along the trajectory. The lower plot illustrates the deformation costs integrated over time for the trajectory indicated above.

be efficiently approached by regression techniques. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ be the set of training examples obtained from n simulations, in which the virtual robot executed n different trajectories $\mathbf{x}_i$ with resulting deformation cost values y_i. Then, the goal is to learn a predictive model $p(y_* \mid \mathbf{x}_*, \mathcal{D})$ for estimating the deformation costs y_* given a new query trajectory $\mathbf{x}_*$.

In theory, arbitrarily curved trajectories through a deformable object can be executed. To bound the complexity of the regression problem, we consider only straight line motions through the object. This is a restriction, but not a strong one, since the trajectories generated by roadmap planners are often piecewise linear motions. The motions considered to estimate the deformation costs are described by a starting point s and end point e on a virtual sphere around the object, both expressed in a spherical coordinate system by an azimuth angle ϕ and an elevation angle θ. Furthermore, we take into account the distance l from the starting point that describes the length of the motion. Figure 5.1 illustrates this parametrization. Thus, $\mathbf{x}_i$ is a five-dimensional vector in our case with $\mathbf{x}_i = [\theta_i^s, \phi_i^s, \theta_i^e, \phi_i^e, l_i]^T$, where the superscript s refers to the starting point and e to the end point. We assume the object's center of mass and correspondingly the center of the sphere to reside in the origin and the radius fixed to a constant r that is determined by the object's dimensions when generating training examples. If we want to determine the deformation costs for arbitrary robot trajectories, it is sufficient to know the rigid body transformation of the object in the world model.

5.3 Modeling Deformation Cost Functions with Gaussian Processes

We approach the problem of estimating the deformation costs of a robot trajectory using the Gaussian process (GP) model (Rasmussen and Williams, 2006). This nonparametric approach to regression specifies a probability distribution over functions. A thorough treatment of Gaussian processes is given by Rasmussen and Williams (2006). In the following, we review Gaussian processes for regression based on their work.

Formally, a Gaussian process is defined as a collection of random variables, any finite subset of which is jointly Gaussian distributed. Therefore, a GP is fully specified by a mean function and a covariance function. The mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ can be defined as:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]\,, \tag{5.2a}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]\,. \tag{5.2b}$$

The Gaussian process can then be written as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))\,. \tag{5.3}$$

In the context of regression, we are usually given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ of samples from a GP. Then, our random variables are the f_i corresponding to the training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. The GP specifies a joint probability distribution over the function values at locations $\mathbf{x}_i$. Therefore, if we define $\mathbf{f}$ to be the vector containing f_i, we can write

$$\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}) , \quad \boldsymbol{\mu} \in \mathbb{R}^n, \mathbf{K} \in \mathbb{R}^{n \times n} . \tag{5.4}$$

with mean $\boldsymbol{\mu}$ and a covariance matrix $\mathbf{K}$.

For notational simplicity, the mean is often assumed to be zero. This is not a strong limitation, since for any deterministic mean function, the data can be rescaled to have an empirical mean of zero; the GP applied to the difference between the observations and the mean function has again zero mean (see Rasmussen and Williams, 2006, Sec. 2.7).

The covariance matrix $[\mathbf{K}]_{ij}$ is specified in terms of a covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$, which encodes prior knowledge about the target distribution, such as smoothness and noise assumptions. It models correlations between the target values in terms of the inputs. Intuitively, it specifies how similar two function values $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ are depending on their input locations $\mathbf{x}_i$ and $\mathbf{x}_j$.

5.3.1 Covariance Functions

In the context of GPs, several covariance functions have been used, each with different properties, thus encoding different assumptions about the functions to be modeled. An overview can be found in Chapter 4 of Rasmussen and Williams' book (2006). In the following, we will discuss two covariance functions in more detail. A popular covariance function applied to a wide range of problems is the squared exponential covariance function. It is given by

$$k_{SE}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{d^2(\mathbf{x}_i, \mathbf{x}_j)}{2\ell^2}\right) . \tag{5.5}$$

Here, ℓ is the characteristic length-scale of the function, and σ_f^2 is the signal variance. These parameters are known as the hyperparameters of the process. This covariance function is stationary, which means the covariance only depends on the distance $d(\mathbf{x}_i, \mathbf{x}_j)$ between inputs, not on their actual position in input space.

New covariance functions can be obtained from the sum and the product of covariance functions. This gives rise to the covariance function with individual length-scales for different input dimensions, thereby implementing automatic relevance determination (Neal, 1996):

$$k_{SE-ARD}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_D \frac{d^2(x_{i,D}, x_{j,D})}{2\ell_D^2}\right). \tag{5.6}$$

Here, ℓ_D are the characteristic length-scales of the individual dimensions D.

For our periodic domain of angular inputs, the Euclidean distance captures their similarity poorly. To correctly consider the distance of the inputs describing points on a sphere, we have to adjust the distance function inside the covariance function. MacKay (1998) describes a nonlinear embedding for periodic inputs to obtain a periodic covariance function: with the mapping $x \mapsto \mathbf{u}(x)$ and $\mathbf{u} = (\cos(x), \sin(x))$, the squared exponential covariance function applied in $\mathbf{u}$-space is given as

$$k_{SE-\pi}(x_i, x_j) = \sigma_f^2 \exp\left(-\left(\frac{2\sin^2\left(\frac{(x_i-x_j)}{2}\right)}{\ell^2}\right)\right). \tag{5.7}$$

The distance between x_i and x_j is expressed in terms of its mapping $\mathbf{u}$ as $d^2(\mathbf{u}_i, \mathbf{u}_j) = (\cos(x_i) - \cos(x_j))^2 + (\sin(x_i) - \sin(x_j))^2 = 4\sin^2\left(\frac{x_i-x_j}{2}\right)$. This two-dimensional mapping describes the distance for points lying on a circle.

In our case, to compute the distance between points on a sphere, we have to consider the input pairs (θ_i, ϕ_i) and (θ_j, ϕ_j) of azimuth ϕ and elevation angle θ. The shortest distance for points on a sphere is given by the great circle distance and can be computed as (Sinnott, 1984):

$$d(\theta_i, \phi_i, \theta_j, \phi_j) = 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\theta_j - \theta_i}{2}\right) + \cos(\theta_i)\cos(\theta_j)\sin^2\left(\frac{\phi_j - \phi_i}{2}\right)}\right). \tag{5.8}$$

To obtain a covariance function that models correlations between trajectories, we incorporate the distance function (5.8) for points on the sphere and determine individual length-scales for starting point, end point, and length:

$$k_{SE-T}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}\left(\frac{d^2(\theta_i^s, \phi_i^s, \theta_j^s, \phi_j^s)}{2\ell_s^2} + \frac{d^2(\theta_i^e, \phi_i^e, \theta_j^e, \phi_j^e)}{2\ell_e^2} + \frac{|l_i - l_j|^2}{2\ell_l^2}\right)\right). \tag{5.9}$$

The squared exponential covariance function is stationary, meaning that the covariance of two inputs only depends on their distance, not on their actual position in input space. This

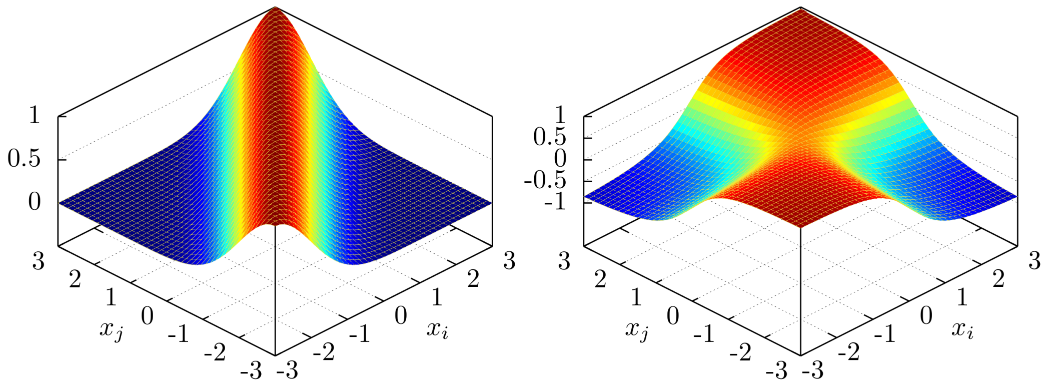


Figure 5.2: The squared exponential (left) vs. the neural network (right) covariance function: Shown are the covariances $k(x_i, x_j)$ of 1-dimensional data points ranging from $[-3, 3]$. While the covariance of data points under k_{SE} rapidly decreases with distance, the covariance of data points under k_{NN} depends on the distance to the origin and saturates with larger distance.

implies a global smoothness assumption, in a sense that the function is assumed to be similarly smooth over the whole input domain. Furthermore, it is infinitely differentiable, which results in very smooth functions and a strong smoothing of data points. In our experiments, we investigate, to what extent these assumptions are justified.

The deformation cost function for trajectories can be non-smooth, there might be discontinuities, for instance, at transitions between trajectories missing an object and trajectories deforming it. Thus, the stationarity and smoothness assumption of the squared exponential kernel might be too strong for our regression problem. Therefore, we additionally consider the nonstationary neural network covariance function, which was described by Neal (1996); Williams (1998, 1999) and corresponds to a neural network with one hidden layer and infinitely many hidden units. It is specified as

$$k_{NN}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \arcsin\left(\frac{\beta + 2\mathbf{x}_i^T \Sigma \mathbf{x}_j}{\sqrt{(1 + \beta + 2\mathbf{x}_i^T \Sigma \mathbf{x}_i)(1 + \beta + 2\mathbf{x}_j^T \Sigma \mathbf{x}_j)}} \right), \tag{5.10}$$

with a bias factor β and $\Sigma = \text{diag}(\ell_1, \ldots, \ell_d)^{-2}$ the length-scale matrix, containing the characteristic length-scales ℓ_D of the individual dimensions D. The signal variance σ_f^2 is as defined above. The neural network covariance function is able to adapt to variations and discontinuities in the data. For instance, it has successfully been used to model discontinuities in large-scale terrain data by Vasudevan et al. (2009). In contrast to the squared exponential covariance function, the neural network covariance of two data points depends on their distance to the coordinate origin, not only on their relative distance. The behavior of the two different covariance functions is illustrated in Figure 5.2.

We show a regression example comparing the two different covariance functions in Figure 5.3. We consider two nonstationary test functions, which have been used for evaluation purposes by DiMatteo et al. (2001), among others. The neural network covariance function is in both cases able to better adapt to the varying smoothness of the functions.

5.3.2 Predictions with the GP model

Given a set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ of training data obtained from the physical simulation engine, we are interested in predicting the target value y_* for a new trajectory specified by $\mathbf{x}_*$. In general, observed data points are subject to noise, that means we do not observe the function values f_i directly, but only noisy values $y_i = f(x_i) + \epsilon$. In the GP model, the noise ϵ is assumed to be independent and identically distributed Gaussian noise with variance σ_n^2. Now, let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n]^\top$ be the $n \times D$ matrix of the n D-dimensional training inputs, $\mathbf{y} = (y_1, \ldots, y_n)$ be the vector of the corresponding target values. In the GP model, any finite set of samples is assumed to be jointly Gaussian distributed, hence we can write the joint probability distribution of the training data and the query $(\mathbf{x}_*, y_*)$

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X},\mathbf{X}) + \sigma_n^2 \mathbf{I} & k(\mathbf{X},\mathbf{x}_*) \\ k(\mathbf{x}_*,\mathbf{X}) & k(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right), \tag{5.11}$$

where $k(\mathbf{X}, \mathbf{X})$ refers to the covariance matrix built by evaluating the covariance function $k(\cdot, \cdot)$ for all pairs of training examples $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Since we assume additive, independent noise, the global noise variance σ_n^2 is only added to the diagonal elements of the covariance matrix. The covariance between test and training samples is given by $k(\mathbf{x}_*, \mathbf{X})$ and $k(\mathbf{X}, \mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{X})^T$. The predictive distribution for the new target value y_* can be obtained by conditioning this $(n + 1)$-dimensional Gaussian distribution on the training data. Applying Bayes' rule and marginalizing over the function values, the predictive distribution $p(y_* \mid \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ for a new trajectory $\mathbf{x}_*$ can be derived, which is again Gaussian with mean

$$\hat{y}_* = k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}\right]^{-1} \mathbf{y} \tag{5.12}$$

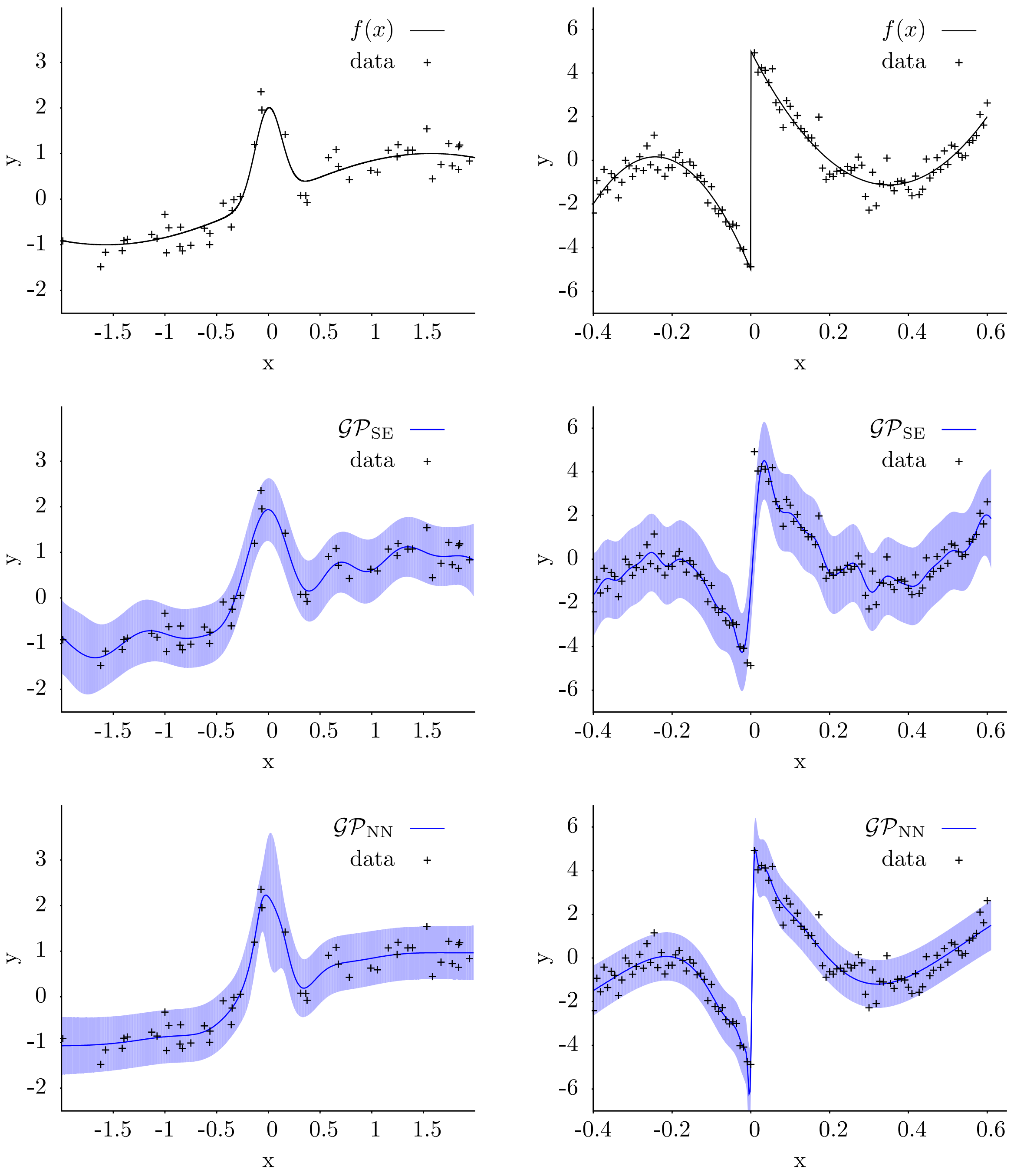


Figure 5.3: GP Regression: we consider two different nonstationary test functions (top row) and the GPs determined from the sampled noisy data points, using the squared exponential (middle row) and the neural network (bottom row) covariance function. Shown are the GP mean functions and the predictive uncertainties in terms of the 2σ-errorbars.

and variance

$$\sigma^2_{\hat{y}_*} = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}\right]^{-1} k(\mathbf{X}, \mathbf{x}_*) + \sigma_n^2 . \tag{5.13}$$

In sum, Eq. (5.12) provides the predictive mean for the deformation costs when carrying out a movement along $\mathbf{x}^*$ and Eq. (5.13) provides the corresponding predictive variance.

For our path planning application, the predictive mean is technically sufficient to give an estimate of the deformation costs that can be expected along a trajectory. However, the predictive uncertainty provided by the GP framework is also useful and could be incorporated into the cost function of the planner. For instance, areas with a higher uncertainty corresponding to less dense sampling of trajectories or a higher variation in the deformation costs could better be avoided.

5.3.3 Learning a GP model

The GP model is a nonparametric model, that means it is represented in terms of the training data points, as can be seen in the equations for the predictive mean and variance above. The covariance function, however, contains different free parameters, such as the length-scale and the signal variance, that have an influence on the characteristics of the process and therefore on the prediction of new data points. To emphasize their role as parameters of a nonparametric model, these are referred to as *hyperparameters* of the process. Learning a GP model thus amounts to determining the hyperparameters $\boldsymbol{\theta}$ of the covariance function that best explain the training data points. In our case, the hyperparameters are given by $\boldsymbol{\theta}_{SE-T} = (\ell_s, \ell_e, \ell_l, \sigma_f, \sigma_n)$ for the squared exponential covariance function and $\boldsymbol{\theta}_{NN} = (\ell_k, \beta, \sigma_f, \sigma_n)$, $k = \{1, \dots, 5\}$ for the neural network covariance function.

The learning problem can be formulated as an optimization problem by finding the hyperparameters $\boldsymbol{\theta}_*$ that maximize the marginal log likelihood of the data (see Rasmussen and Williams, 2006, Chap. 5.4): $\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y} \mid X, \boldsymbol{\theta})$ with

$$\log p(\mathbf{y} \mid X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T K^{-1}\mathbf{y} - \frac{1}{2}\log|K| - \frac{n}{2}\log(2\pi) . \tag{5.14}$$

The different terms in this objective function can be interpreted as follows (Rasmussen and Williams, 2006, Chap. 5.4): the first term contains the target values and therefore measures the data fit. The second term only depends on the covariance function and penalizes complex models. The last term is a normalization constant. By trading the data fit against the model complexity in this problem formulation, Occam's razor principle is automatically implemented. Thus, finding the simplest model that explains the data is encouraged.

To solve this optimization problem for a given data set, we use a standard conjugate gradient optimization approach. This requires the partial derivatives of the log likelihood function with respect to the hyperparameters θ_i, which can be computed as (see Rasmussen and Williams, 2006, Chap. 5.4):

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\log p(\mathbf{y} \mid X, \boldsymbol{\theta}) &= \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial\theta_i}K^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_i}\right)\\ &= \frac{1}{2}\mathrm{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - K^{-1})\frac{\partial K}{\partial\theta_i}\right) \qquad \text{with } \boldsymbol{\alpha} = K^{-1}\mathbf{y}\,. \end{aligned} \tag{5.15}$$

The complexity of the optimization is in $O(n^3)$, since the covariance matrix K needs to be inverted in each iteration of the optimization.

5.4 Efficient Regression by Problem Decomposition

The GP model introduced above allows us to predict the expected deformation costs of a new trajectory based on a set of training samples. In high-dimensional input domains, such as our trajectories in 3D space, a considerable set of training samples is needed to obtain a good function approximation. In this nonparametric approach to regression, the function is entirely represented in terms of the training data points: training the GP model as well as computing the predictive distribution for a new data point has a runtime cubic in the number of training samples, due to the necessary inversions of the covariance matrix (cf. Eqs. (5.12), (5.13), (5.14)). For data sets consisting of thousands of training samples, the approach thus becomes inefficient.

Various techniques have been proposed that address scalability in GPs and are able to deal with large data sets. For example, sparse approximations to Gaussian processes represent the data distribution by a small number of input points only. These points can be chosen from the training set such that they maximize some information criterion about the posterior distribution (Lawrence et al., 2003; Seeger et al., 2003). This can lead to problems with the optimization of the hyperparameters, as Snelson and Ghahramani (2006) observed,

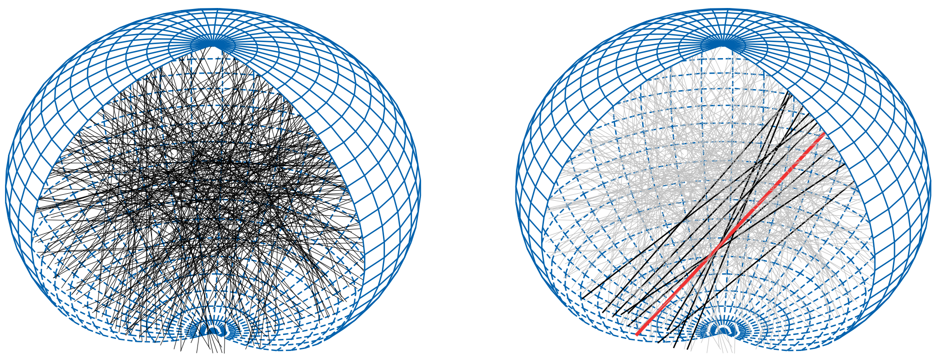

(a) Preprocessing (b) Local Approximation

Figure 5.4: Local approximation idea: (a) A large data set of 3D training trajectories is generated. (b) When predicting the deformation costs of a new trajectory (red), we determine its nearest neighbors (black) and build a local GP to compute the predictive distribution of the new target value.

since selection of informative inputs and adaptation of hyperparameters interfere with each other. To overcome this problem, they introduced a sparse GP framework, which represents the predictive distribution by a set of pseudo-inputs. These are treated as additional hyperparameters of the process and determined in the optimization step to maximize the data likelihood. In both approaches, however, GP training and selecting inputs becomes more involved.

Vasudevan et al. (2009) suggest a different strategy to handle large data sets. They use the GP framework to generate probabilistic terrain models from 3D point cloud data acquired with a laser scanner. In this context, millions of data points are available for regression and need to be handled. The idea of their approach is to organize the data points in a k-d tree, which supports fast nearest neighbor search, and to use only a subset of the data for training the hyperparameters of the GP as well as for regression. When actually computing the predictive distribution for a new data point, they retrieve its nearest neighbors from the k-d tree to set up a local GP. Since the number of inputs in this local GP is small, inference can be carried out efficiently.

Inspired by the approach of Vasudevan et al. (2009), we decompose our regression problem into a number of local ones. When predicting the deformation costs of a query trajectory $\mathbf{x}_*$, we determine its M nearest neighbors from the training data set as

$$\mathbf{X}'(\mathbf{x}_*) = [\mathbf{x}'_1; \dots; \mathbf{x}'_M] = \underset{[\mathbf{x}'_1;\dots;\mathbf{x}'_M]}{\operatorname{argmin}} \sum_{k=1}^{M} d_T(\mathbf{x}'_k, \mathbf{x}_*). \tag{5.16}$$

Similar to our covariance function from Eq. (5.9), the distance function $d_T(\cdot,\cdot)$ for trajectory samples considers the great circle distance on the surface of the sphere for starting and end points of the trajectories (Eq. (5.8)) and can be computed as

$$d_T(\mathbf{x}_i, \mathbf{x}_j) = \|l_i - l_j\| + d(\theta_i^s, \phi_i^s, \theta_j^s, \phi_j^s) + d(\theta_i^e, \phi_i^e, \theta_j^e, \phi_j^e)\,. \tag{5.17}$$

The M nearest neighbors to a query trajectory $\mathbf{x}_*$ can be determined efficiently using a k-d tree (Bentley, 1975; Preparata and Shamos, 1993). This data structure is built once from the training data in $O(n \log n)$ for n training data points. Then, searching for the nearest neighbors can be carried out in $O(\log n)$ for n training data points. This reduces the runtime for regression from $O(n^3)$ to $O(\log nM^3)$ with $M \ll n$.

The M closest neighbors $\mathbf{X}'$ to the query trajectory $\mathbf{x}_*$ are the training data points with the highest influence on the prediction of y_* in the GP framework. Considering only $\mathbf{X}'$ instead of $\mathbf{X}$ in the GP is equivalent to assuming that $k(\mathbf{x}_*, \mathbf{x}_i) = 0$ for all $\mathbf{x}_i$ that are excluded from $\mathbf{X}'$. We illustrate this line of action in an example in Figure 5.4: we generate a large data set of training trajectories using the simulation engine, and when determining the deformation cost of a new trajectory, we retrieve its nearest neighbors from the k-d tree to build a local GP.

One remaining problem we need to address is the question of how to determine the hyperparameters for the local GPs. Different strategies can be considered: As described in the work of Vasudevan et al. (2009), the hyperparameters can be determined for a subset of the available set of training data and applied to the local GPs. Another possibility is to optimize the hyperparameters for each local GP individually. This strategy implements a form of nonstationarity for the squared-exponential covariance function, as the length-scales of the GPs can be adapted to the local structure of the function. In our experiments, we evaluate both optimization strategies with respect to runtime and prediction accuracy.

5.5 Experimental Results

In this section, we present evaluations of our approach to model the deformation cost functions of objects with Gaussian processes. Using the simulation framework described in Chapter 3, we generated several data sets consisting of trajectory samples that potentially lead to object deformations for artificial and real deformable objects. We considered trajectories in 2D that describe the motions of a wheeled robot and trajectories in 3D that describe the movements of a manipulation robot end effector. To generate random trajectory samples, we proceeded as follows: We first computed a set of uniformly distributed starting points and end points. In the case of 2D trajectories, we considered points on a bounding circle around the object with an angular resolution of 1°. This sampling results in a distance of approximately 10 cm between neighboring points. In the case of 3D trajectories, we sampled starting point and end point on a bounding sphere around the object. To compute sample points on the sphere that are distributed uniformly and equidistantly, we used the spiral point algorithm by Rakhmanov et al. (1994). We considered a set of 5,473 points with a minimum distance of approximately 10 cm between neighboring points. A trajectory sample can then be drawn by randomly choosing starting point and end point from the generated point set. Using a fast collision checking routine, we discard trajectories that do not lead to object deformations, since for these trajectories, the deformation costs naturally amount to zero. If a collision is detected, we carry out an accurate physical simulation of the robot movement to determine its deformation costs. An overview of the generated data sets is given in Table 5.1. In addition to performing evaluations on the generated data sets, we use them in the planning applications in the next chapter.

In all our experiments, we evaluate the accuracy of the predictions using the mean absolute error (MAE). To obtain a measure that is independent of the scale of the distribution, we furthermore consider the standardized mean squared error (sMSE) that is normalized by the variance σ^2_{test} of the test set and is computed as

$$\text{sMSE} = \frac{1}{n} \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{\sigma^2_{\text{test}}} . \tag{5.18}$$

For a trivial model, a Gaussian distribution with mean and variance of the training set distribution, which always predicts the mean of the training distribution regardless of the input, this error measure is approximately one.

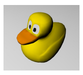
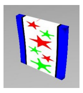
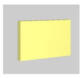
Data set		# trajectories	# tetras	runtime
Artificial objects:				
	D1: Duck (2D)	4,284	530	7 h
	D2: Curtain-A (2D)	4,693	500	10 h
Real objects:				
	D3: Curtain-R (2D)	2,035	285	5.5 h
	D4: Foam (3D)	22,950	385	24 h
	D5: Teddy (3D)	12,620	940	24 h

Table 5.1: Data sets for different deformable objects: we generated a set of example trajectories leading to deformations using the simulator. The simulation time depends on the length of the trajectories and on the number of elements of an object, it increases for more complex objects.

The sMSE and MAE error losses only take into account the predictive mean of the model. Since the Gaussian process framework provides us with an estimate of the uncertainty of a prediction, we evaluate the fit of this predictive distribution by considering the negative log predictive density (NLDP) of the true targets. This loss function is commonly used to evaluate predictive distributions (cf. Rasmussen and Williams, 2006, Chapter 2) and can be computed as:

$$-\log p(y_* \mid \mathbf{x}_*, \mathcal{D}) = \frac{1}{2}\log(2\pi\sigma_*^2) + \frac{(y_* - \hat{y}_*)^2}{2\sigma_*^2} . \tag{5.19}$$

It is minimal when the variance equals the error and penalizes both over-confident and under-confident estimates. This loss can be standardized by subtracting the NLPD of the trivial model, that is a Gaussian distribution with mean and standard deviation of the training set distribution. Thus, we compute the mean standardized log loss (MSLL) as

$$\text{MSLL} = \frac{-1}{n}\sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i, \mathcal{D}) - \log p_{\text{trivial}}(y_i \mid \mathbf{x}_i, \mathcal{D}) . \tag{5.20}$$

This loss function is approximately zero for simple models and negative for better models.

We investigate the effects of different optimization strategies and parameters. In particular, we compare two different covariance functions for the modeling task, the squared exponential covariance function introduced in Eq. (5.9) and the nonstationary neural network covariance function from Eq. (5.10). We analyze the required number of training samples as well as the number of nearest neighbors to be considered for the individual prediction tasks. To demonstrate the benefits of decomposition and GP regression, we compare our approach to a full GP model using all training points and to a weighted average M-nearest neighbor strategy. To summarize, the different strategies we evaluate are:

GPD our GP-Decomposition strategy, for each test point, the M nearest neighbors are selected from the N training points to build a local GP for regression. The hyperparameters, used in all local GPs, are optimized once on a subset of all trajectories. We consider the squared exponential (GPD-SE) and the neural network (GPD-NN) covariance function.

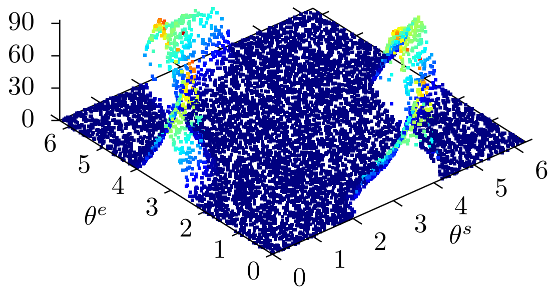
GPO GP decomposition and local Optimization strategy, in addition to building local GPs for regression of test samples, the hyperparameters are optimized for each local GP and test point individually on $M \ll N$ trajectories. We consider the squared exponential (GPO-SE) and the neural network (GPO-NN) covariance function.

GPF a GP-Full model using all available training trajectories (if computationally tractable) for hyperparameter optimization and prediction of a test point, with the squared exponential (GPF-SE) or the neural network (GPF-NN) covariance function.

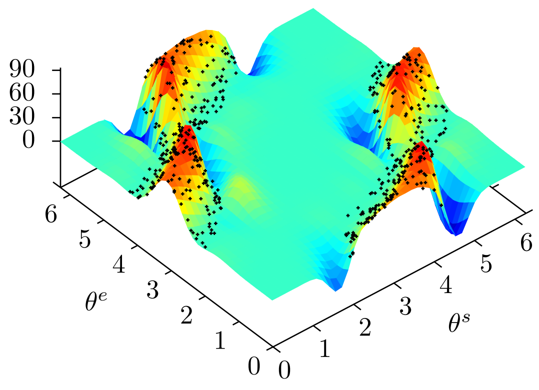
IDW Inverse Distance Weighting, the baseline strategy predicts the weighted average of the M nearest neighbors to a test point, with weights corresponding to the inverse of the distance to a test point.

5.5.1 Deformation Cost Function Example

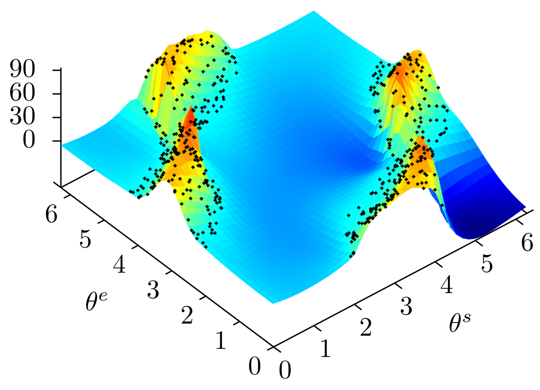
As an illustrative example, we consider the deformation cost function for the curtain that is deformed by the wheeled robot. The sampled deformation cost function is shown in Figure 5.5, together with learned GP models using the neural network and squared exponential covariance function. From this example, we can already gain some insight into the general behavior of the deformation cost function. It illustrates that the deformation cost function is zero in large areas corresponding to trajectories that do not lead to contact with the ob-



(a) Deformation cost function



(b) $\mathcal{GP}_{\text{SE}}$



(c) $\mathcal{GP}_{\text{NN}}$

Figure 5.5: The deformation cost function for a curtain that is deformed by a robot navigating in two dimensions. The trajectories are parametrized by starting point angle θ_s and end point angle θ_e, the length of the trajectory is assumed to be fixed such that the robot passes the curtain.

ject. When learning the GP model, we therefore only consider samples that actually lead to object deformations, otherwise, the zeroes are overrepresented and the function is not adequately modeled. The trajectories with zero deformation costs could still be considered when building local GPs. In our experiments, however, we ignore them.

The deformation cost function seems to be nonstationary, zero in large parts, and rapidly increasing from trajectories barely touching the curtain to trajectories leading to large deformations, in particular, as the curtain is a very thin object. The squared exponential kernel adapts to the rapidly changing function values by choosing very small length-scales. This, however, leads to oscillations in less densely sampled areas. For instance, in the border regions, the costs drop below zero. The neural network kernel models this nonstationary function in a better way. It obtains rather smooth estimates of zero even in regions with no samples, except in one region, in which the estimated deformation costs drop below zero. In general, it models the data points well. In the following, we investigate the modeling properties of both kernels in more detail.

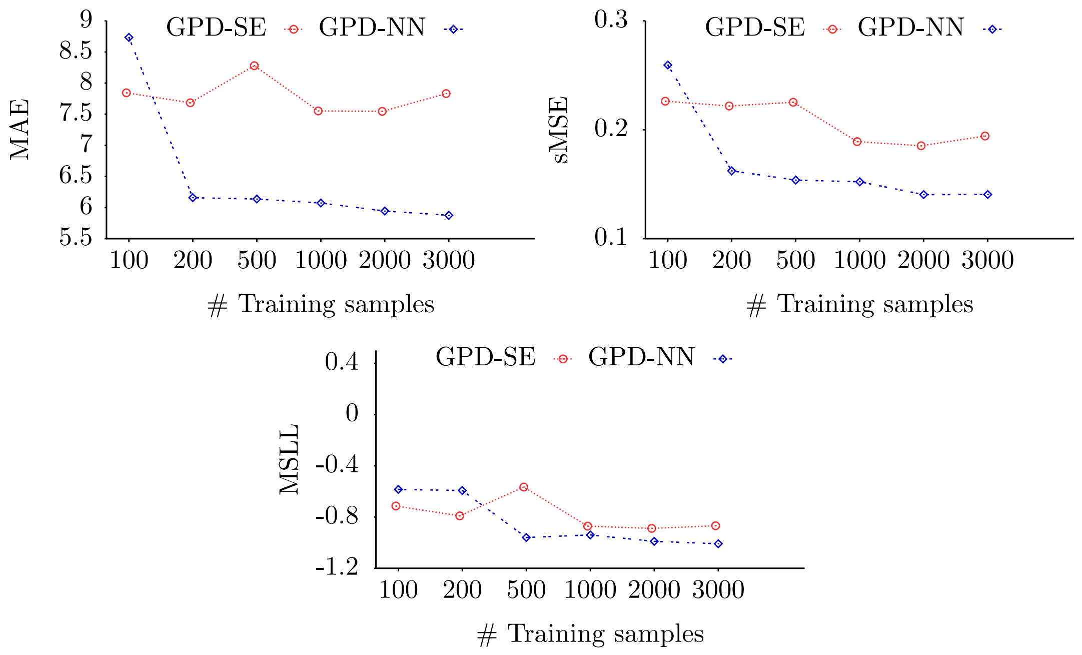


Figure 5.6: Prediction accuracy of the local GPD models depending on the number of samples used to train the GP hyperparameters. Data set: 3D-Teddy, 5,000 training trajectories were available and the 50 nearest neighbors were used to build the local GPs.

5.5.2 GP Training and Number of Training Samples

GP training is the process of optimizing the hyperparameters of the covariance function such that the model best fits the data. In the first set of experiments, we investigated how many training samples are required to train GP models and to obtain accurate predictions.

With the GPD strategies, only a subset of the available training data set is used to train the hyperparameters of the GP models. In a first experiment, we investigated the effect of the number of samples N used to train the hyperparameters on the prediction accuracy. We considered the Teddy-3D data set and randomly selected 10,000 trajectories as training data and 300 trajectories as test data. For a fixed number of 25 nearest neighbors to build the local GPs, we evaluated the prediction errors (MAE and sMSE) as well as the MSLL for varying N. The results of this experiment are shown in Figure 5.6 and illustrate that training sets larger than 1,000 samples do not lead to improved accuracies. The computation time for hyperparameter optimization, though, is cubic in the number of samples. For 1,000 samples, optimization with the squared exponential covariance function requires up to two minutes, with the neural network covariance function around five minutes due to more involved covariance computations. For 3,000 samples, the optimization already takes up to half an hour in case of the squared exponential and up to one hour in case of the neural network covariance function. In the following experiments, we therefore limit the maximum number of samples to 1,000 when learning the hyperparameters of the GP models.

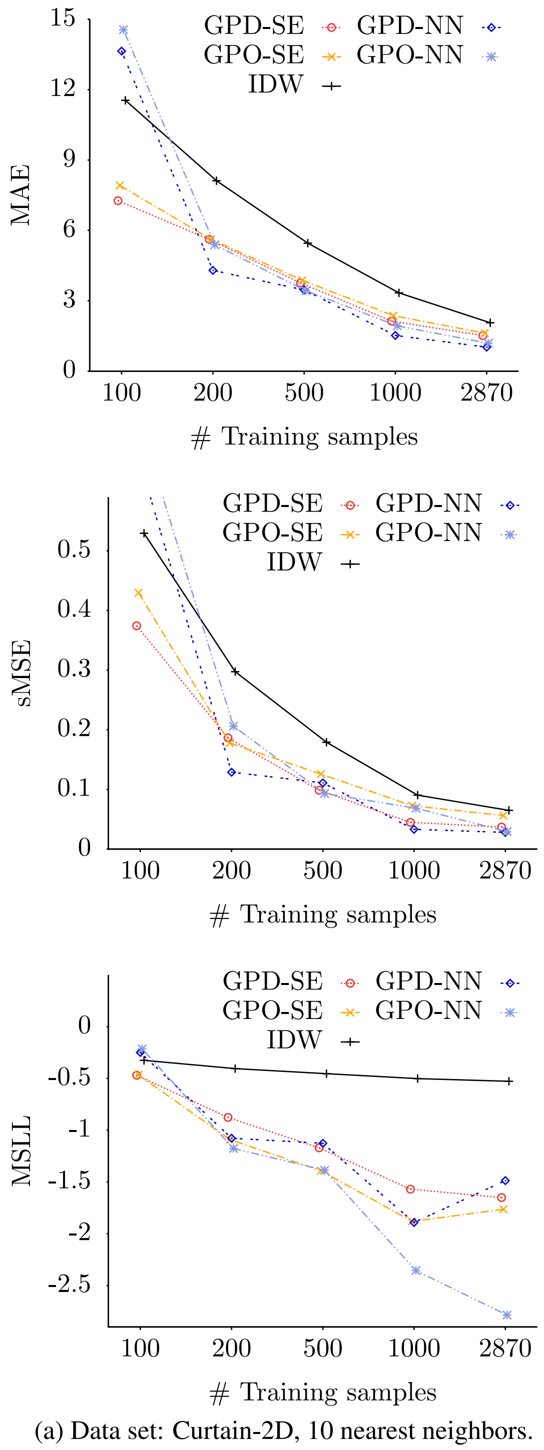
We additionally investigated the required number of trajectory samples to obtain accurate predictions for the deformation costs. We split the data sets into 80 % training trajectories and 20 % test trajectories. From the available set of training trajectories, a varying number of samples was used for regression of the test samples. In this experiment, the number of nearest neighbors used to build local GPs was fixed to ten. The results of this experiment for all considered strategies are summarized in Figure 5.7a for the 2D curtain data set and Figure 5.7b for the 3D foam data set. Increasing the number of training samples obviously leads to a smaller prediction error for all considered strategies. In 2D, considering 1,000 samples leads to good results and the accuracy does not improve significantly for larger datasets. In 3D, the number of input dimensions is larger, thus more training samples are required. The errors for the GP models are in general smaller compared to the baseline strategy, in particular, if fewer training samples are available. For large training data sets, however, the error of the baseline strategy approaches the error of the GP models. As generating new training samples by means of simulations is time-consuming, the training data sets can not be arbitrarily increased and the GP models allow for a better trade-off between the size of the training set and accuracy. For the considered number of nearest neighbors, we cannot observe a significant difference between squared exponential and neural network kernel. Locally optimizing the hyperparameters does not seem to improve the prediction accuracy either, at least for the considered number of training samples for the local GPs in this experiment. The MSLL, however, is smaller for the locally optimized GPs, which indicates more accurate uncertainty estimates for these strategies.

5.5.3 Number of Nearest Neighbors

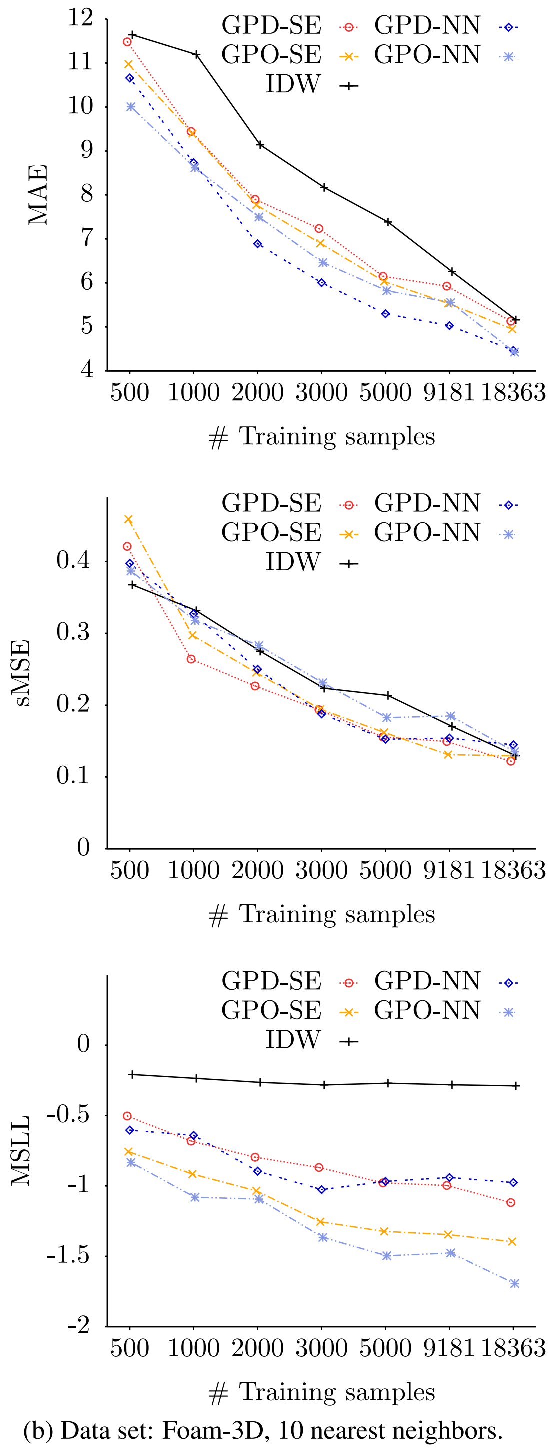
In a further experiment, we investigated the influence of the number of nearest neighbors on the prediction accuracy. For a fixed number of 1,000 training samples for the 2D data sets and 2,000 training samples for the 3D data sets, we evaluated the prediction error when considering up to 100 nearest neighbors for each prediction task. The results of this experiment are summarized in Figure 5.8a for the 2D curtain data set and Figure 5.8b for the 3D foam data set. As the prediction error constantly increases for the weighted average nearest-neighbor strategy, it is not included here. Increasing the number of nearest neighbors leads to more accurate predictions and uncertainty estimates for all GP models. In the 2D data sets, however, this effect is not that pronounced. With 50 nearest neighbors, the performance is comparable to a full GP model considering all data points. Using only the M nearest neighbors when evaluating the GP model, however, speeds up computation time, since no computations with huge matrices are required. In case of the neural network kernel, evaluation of one test sample requires approximately 20 ms for a GP with 50 data points, in contrast to 550 ms for a GP with 2,000 data points. Locally optimizing the hyperparameters does not notably influence the prediction errors, the uncertainty estimates, however, are more accurate. If the hyperparameters are optimized, a computational overhead of 200 ms per sample is introduced. In contrast to a full GP model, the local approximation strategies can deal with even larger data sets, thus resulting in more accurate predictions. The experiments indicate, that a number of 50 nearest neighbors leads to similar results to the full GP model while at the same time significantly reducing computation time.

5.5.4 Modeling Uncertainty

The GP framework provides us with an estimate of the uncertainty for a predicted function value. In addition to the prediction accuracy, we investigated the reliability of the uncertainty estimates for the different GP models in the above experiments in terms of the MSLL. We illustrate the ability of the different strategies to model the uncertainty in an example. Figures 5.9 and 5.10 show the prediction errors and the corresponding uncertainty estimates in terms of 2σ-error bounds for the individual test samples of a 2D and a 3D data set, respectively. Furthermore, diagonal plots visualize, to what extent the predicted values deviate from the true values. For the baseline strategy, we approximated the uncertainty with the variance of the training data set. All GP strategies are able to more accurately take account of the uncertainty than the baseline strategy. When comparing the different covariance functions, the neural network covariance function leads to smaller errors and uncertainty estimates for most samples but also contains some more severe outliers than the squared exponential covariance function. The locally optimized GPs further improve the uncertainty estimate.

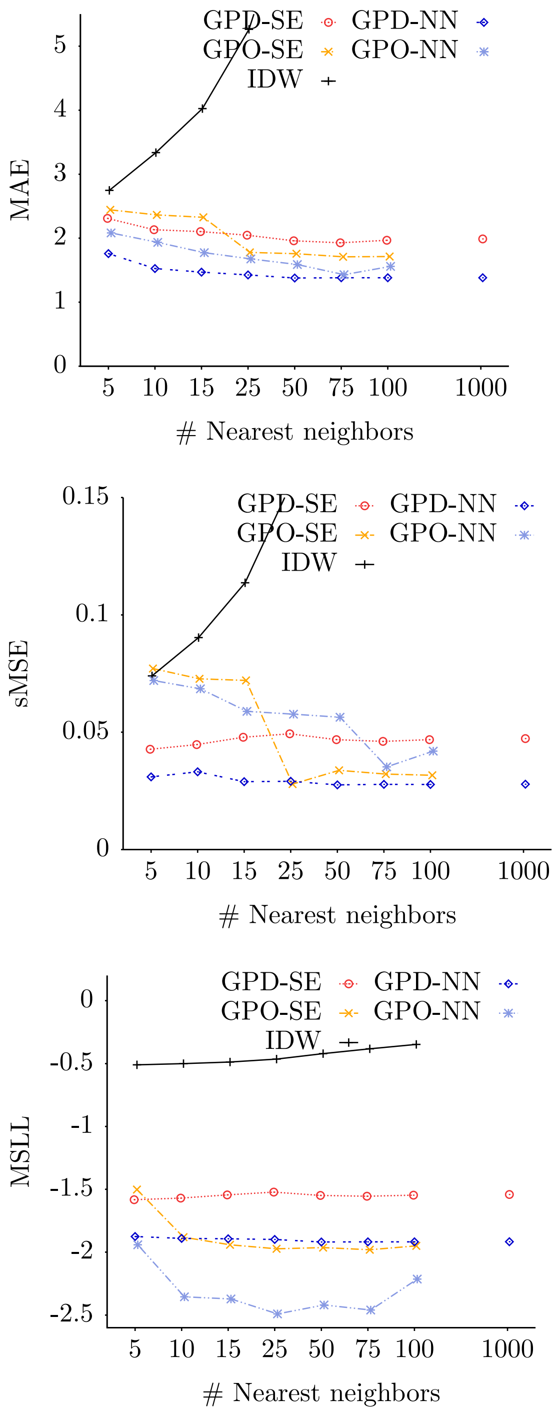


(a) Data set: Curtain-2D, 10 nearest neighbors.

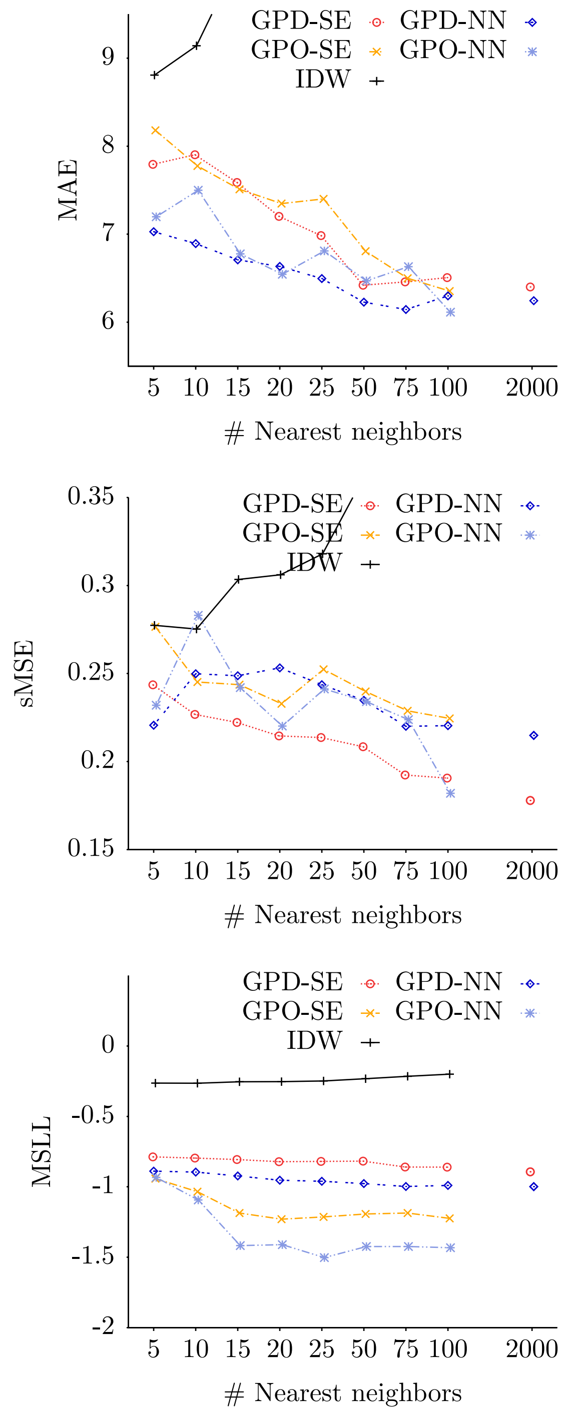


(b) Data set: Foam-3D, 10 nearest neighbors.

Figure 5.7: Comparison of different strategies to predict the deformation costs of a robot trajectory: shown are the MAE, sMSE, MSLL depending on the number of training samples.



(a) Data set: Curtain-2D, 1,000 training samples.



(b) Data set: Foam-3D, 2,000 training samples.

Figure 5.8: Comparison of different strategies to predict the deformation costs of a robot trajectory: shown are the MAE, sMSE, MSLL depending on the number of nearest neighbors. The full GP models correspond to 1,000 and 2,000 nearest neighbors, respectively.

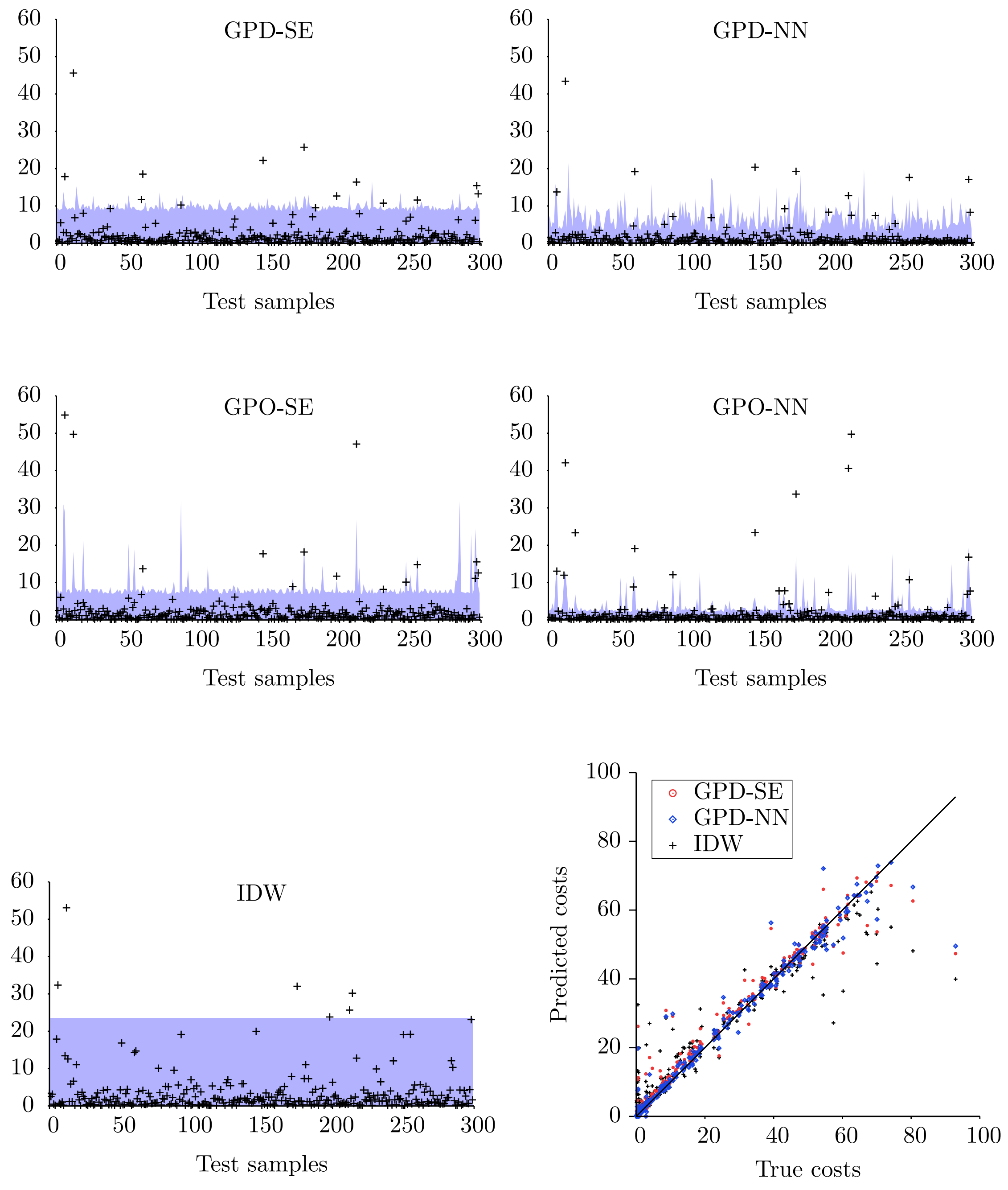


Figure 5.9: Uncertainty estimates for the different models. Shown are the absolute errors and the estimated uncertainties for each test sample. Data set: Curtain-2D, 1,000 training samples, 10 nearest neighbors.

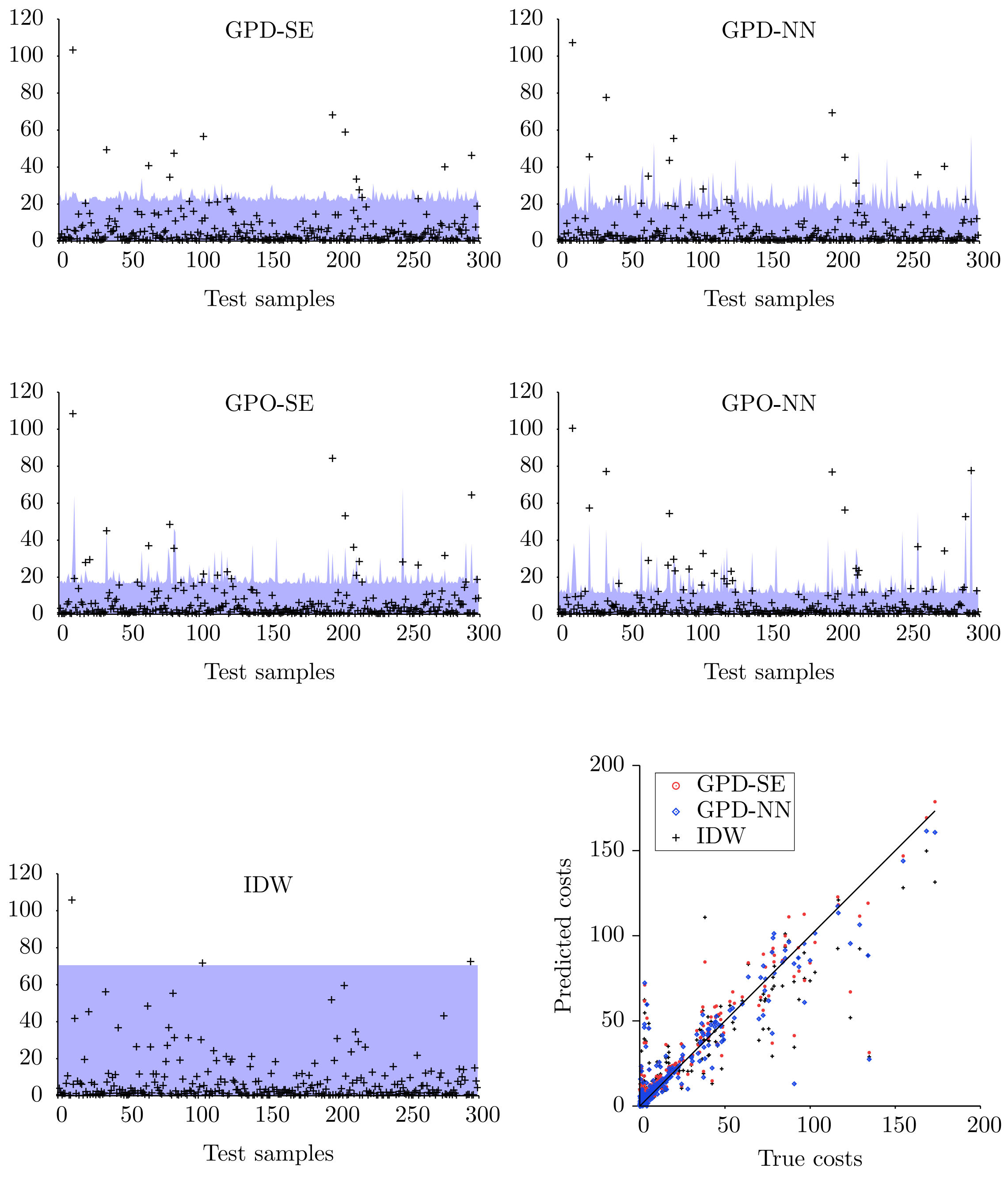


Figure 5.10: Uncertainty estimates for the different models. Shown are the absolute errors and the estimated uncertainties for each test sample. Data set: Foam-3D, 5,000 training samples, 10 nearest neighbors.

5.5.5 Statistical Evaluation

The above experiments showed that a number of 1,000 training samples leads to good prediction results in the case of 2D trajectories. For 3D trajectories, naturally a higher number of training examples is required, but we already obtain decent predictions for 5,000 training samples. For the 3D data sets, a number of 50 nearest neighbors for building local GPs seems to be a reasonable choice both with respect to minimizing the prediction errors and the MSLL. In 2D, we set the number of nearest neighbors to 25, since more nearest neighbors do not lead to improved prediction results.

With these parameters identified, we performed a 10-fold cross-validation on all data sets to obtain statistical data on the performance of the different strategies. Furthermore, we compared the GP models to the baseline strategy, which predicts the weighted average over the ten nearest neighbors. We split the available trajectories into ten folds, and in each run, the test samples were randomly chosen from one fold and the training samples were randomly chosen from the remaining nine folds. The results of this experiment are summarized in Figure 5.11 for the different strategies and data sets we considered. In terms of the MAE, the strategies using the neural network covariance function significantly outperform both the baseline and the squared exponential covariance function. For the sMSE, the difference is less pronounced, but still, the neural network covariance function leads to the smallest overall errors. The uncertainty estimates of the two different covariance functions are comparable, and in most cases, they are improved when locally optimizing the hyperparameters.

5.5.6 Computation Time

In the cross-evaluation experiment above, we analyzed the runtime of the different strategies. For the squared exponential covariance function, GP hyperparameter training for 1,000 samples is in the order of minutes, for 3,000 samples, the optimization already takes up to half an hour. For the neural network covariance function, the computation time ranges from around five minutes for 1,000 samples to approximately one hour for 3,000 samples. Evaluation of a full GP model with 1,000 samples requires 17 ms for the squared exponential and 83 ms for the neural network covariance function to predict one test sample. When considering only 50 samples in the local GPs, evaluation of one sample takes 3 ms for the GPD-SE strategy and 23 ms for the GPD-NN strategy. If the hyperparameters are adapted individually, the GPO-SE strategy requires 30 ms and the GPO-NN strategy requires 230 ms

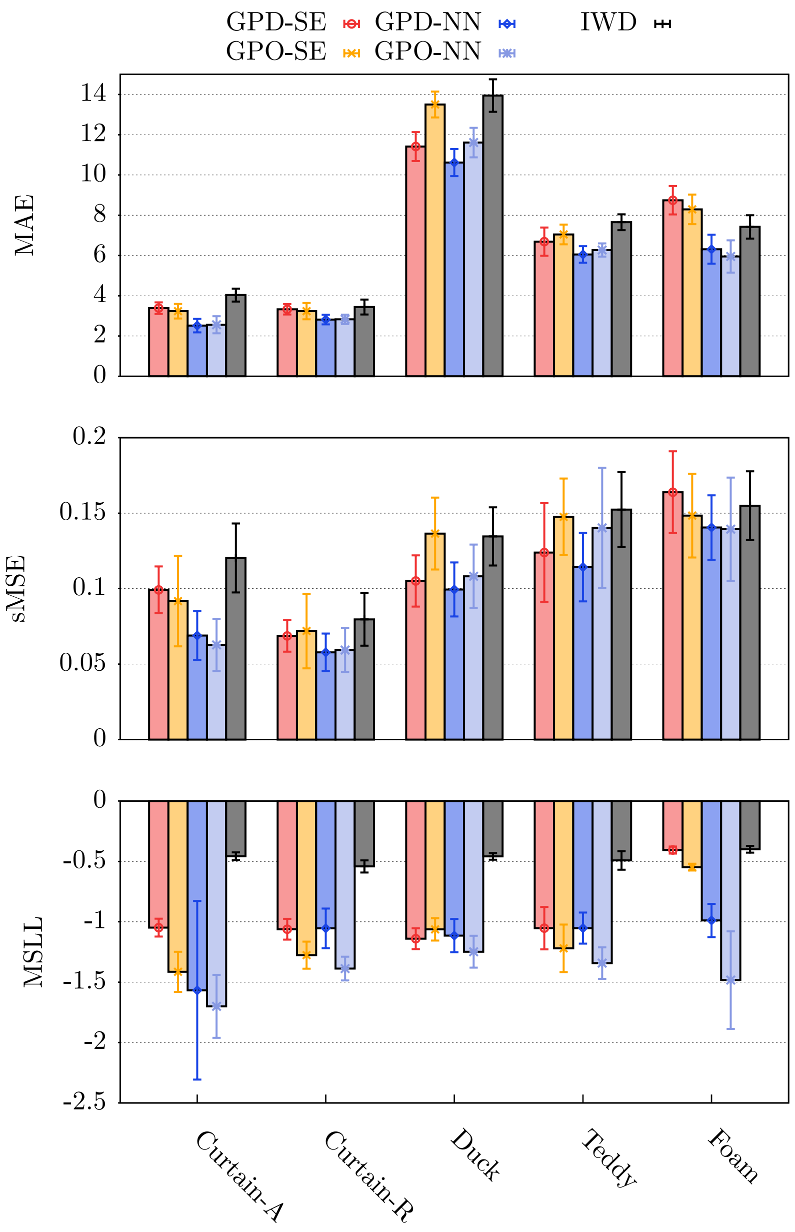


Figure 5.11: Comparison of the different strategies to predict the deformation costs of trajectories in a ten-fold cross-evaluation.

for the prediction of one sample. Thus, considering the results from Section 5.5.5, the GPD strategies allow for the best trade-off between prediction accuracy and runtime.

5.6 Summary

In this chapter, we discussed how to consider the costs of object deformations when planning robot motions. We introduced a measure for the deformation costs that is based on the deformation energy of the affected objects along the robot trajectory. These deformation costs can be computed using the simulation framework discussed in Chapter 3, given appropriate deformation models of the obstacles in the environment, as determined, for instance, with the approach presented in Chapter 4.

To avoid time-consuming simulations during path search, we introduced the concept of object deformation cost functions. By assuming stationary objects that can be deformed but cannot be moved by the robot, we can precompute the deformation cost function for each object. We generate a set of trajectory samples leading to object deformations in preliminary simulations and realize a deformation cost model for each object that is based on nonparametric Gaussian process regression. The Gaussian process models a distribution over all possible deformation cost functions based on a set of training trajectories and can be evaluated efficiently when searching for a path to a goal. In addition to providing an estimate of the deformation costs of trajectories, the Gaussian process model is based on a sound probabilistic framework that allows us to incorporate and model uncertainties in the data.

In the next chapter, we present our motion planner that incorporates the deformation cost functions of objects and thus is able to trade off travel costs of the robot against the costs of potential object deformations.

6 Motion Planning for Real Robots

In this chapter, we combine the learned deformation models from Chapter 4 and their corresponding cost functions introduced in Chapter 5 into a unified motion planning framework that is able to consider deformable objects. We first discuss the integration of the components into a general probabilistic roadmap-based planning framework before presenting two implementations on different robotic platforms. We show how to plan motions for a manipulation robot operating in a three-dimensional workspace as a first application of the planner. Subsequently, we present an implementation for a wheeled robot that navigates in a two-dimensional workspace.

An issue that is raised for robots operating in real-world environments is the problem of collision avoidance. Dynamic and unforeseen obstacles not contained within the map can appear while the robot is executing a motion. Obstacles blocking a path need to be detected in time to avoid damage to the robot as well as to the unexpected obstacle. This is of particular relevance in environments shared between robots and humans.

Traditionally, sensor-based collision-avoidance techniques adapt the motions of the robot to avoid unforeseen and dynamic obstacles. In our case, however, such an approach would not always lead to the desired behavior. For instance, if the planner chooses a path passing through a curtain, the robot will inevitably be in contact with the curtain. A collision avoidance behavior is not desirable in this situation as we explicitly allow for contacts with deformable objects. But consider a person appearing behind the curtain. Then, the robot needs to stop to avoid a collision with the person. Thus, when executing a path in our application, the robot needs to be able to distinguish between *allowed* contacts with deformable obstacles and collisions with unforeseen obstacles that must be avoided in any case. We will address this problem and present an approach that allows the robot to interpret its sensor measurements accordingly during path execution.

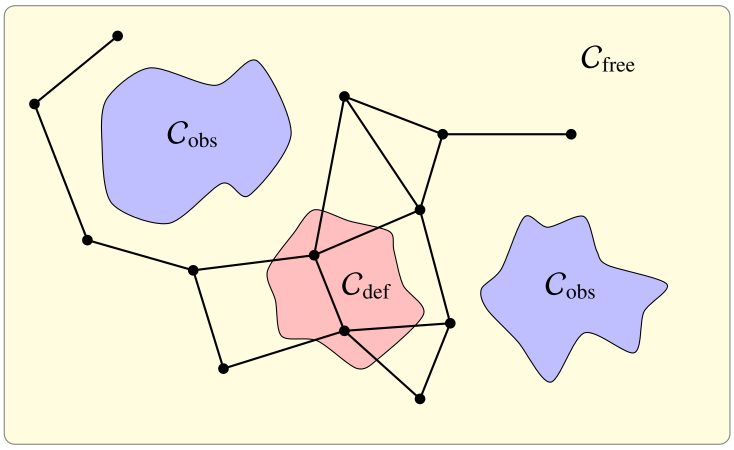


Figure 6.1: Key idea of probabilistic roadmaps: a graph represents the configuration space of the robot. Nodes and edges are added to the graph if they are in C_{free}, the free configuration space. Our planner also accepts samples in C_{def}, leading to collisions with deformable obstacles.

6.1 The Motion Planning Framework

The motion planning problem is defined as finding a continuous path from an initial robot configuration to a goal configuration that satisfies certain constraints; given a model of the environment and the obstacles, the path is in general required to be collision-free. To plan trajectories for our robots, we use the probabilistic roadmap framework (PRM), which has originally been introduced by Kavraki et al. (1996). This planning approach represents the continuous configuration space of robots by a discrete set of samples, and thus allows for the application of discrete search methods. It has gained large popularity since it is able to represent high-dimensional configuration spaces. Hence, it has been applied to a variety of motion planning problems, from robotics, over manufacturing, to biological studies, for instance, in the context of protein-folding.

The key idea of the probabilistic roadmap approach is to represent the collision-free configuration space C_{free} of the robot by a set of samples that form the nodes of a graph. Edges in this graph describe feasible trajectories between neighboring configurations. Figure 6.1 illustrates this representation. We can compute such a roadmap given a geometrical model of the environment and a collision-detection module. The roadmap is constructed by generating a set of random samples of robot configurations and inserting them into the roadmap, if no collision with a static obstacle is reported. After a reasonable number of samples covering C_{free} has been generated, a local planner tries to connect neighboring nodes with simple

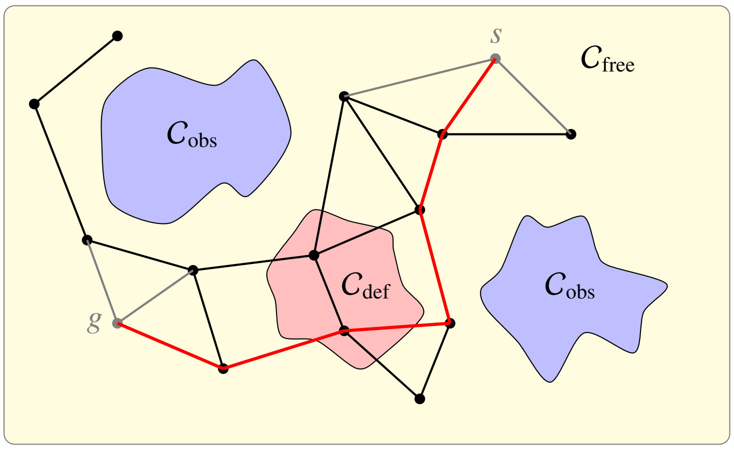


Figure 6.2: Path query: after connecting starting point s and goal point g to the roadmap (grey), A* finds the path with the best trade-off between travel costs and deformation costs (red).

paths, for instance, with straight-line motions. An edge is inserted if the path is feasible, that means if it does not lead to collisions with obstacles. This again can be verified by collision checks for densely sampled robot configurations along the path.

The efficiency of the roadmap planner relies on the availability of a fast collision-detection routine, as thousands of collision-checks need to be carried out when computing a roadmap. We discussed different efficient collision-detection algorithms in Section 3.1.1. Since our planner is integrated into the simulation framework discussed in Chapter 3, a natural choice for our application is the spatial hashing approach integrated into the simulation framework to detect collisions between the robot and the obstacles in the environment.

When planning a trajectory to a specific goal, the first step is to connect the initial and goal configuration to the roadmap. If the connection step is successful, a path can be extracted from the roadmap by applying a standard graph search technique, such as Dijkstra's algorithm (1959) or A* (Hart et al., 1968), Figure 6.2 illustrates a path query in the roadmap. The mentioned graph search techniques are optimal, they find a least-cost path in the roadmap, if one exists. This, in turn requires the definition of a cost function. Most motion planning applications assign costs to the edges in the graph that correspond to the distance of their nodes in configuration or workspace.

6.1.1 Path Costs

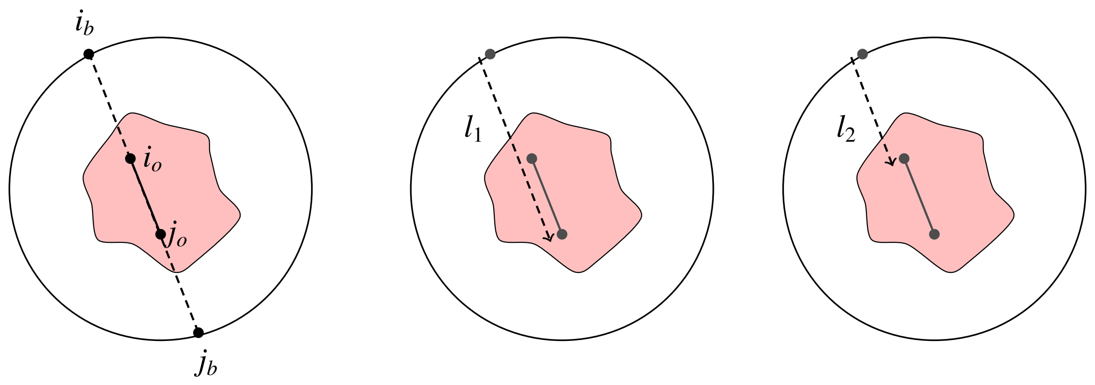
In our application, the robot is allowed to interact with deformable objects. To account for that when planning its motions, we include samples and edges leading to collisions with these objects when generating the probabilistic roadmap (see Figure 6.1). Thus, the configuration space represented in our roadmap is $C_{\text{free}} \cup C_{\text{def}}$. Accordingly, when planning trajectories, we need to consider the costs of deforming objects along an edge. Our planner uses a weighted sum between the travel costs in configuration space and the deformation costs. The costs for an edge between the nodes i and j given by

$$C(i, j) := \alpha\, C_{def}(i, j) + (1 - \alpha)\, C_{travel}(i, j), \tag{6.1}$$

where $\alpha \in [0, 1]$ is a user-defined weighting coefficient. The term $C_{def}(i, j)$ represents the costs that are introduced when the robot deforms objects along its trajectory and $C_{travel}(i, j)$ corresponds to the distance between nodes in configuration space. With this cost function, we use A* to search for the path with the optimal trade-off between deformation costs and travel costs. We use the distance to the goal configuration weighted with $(1 - \alpha)$ as heuristic. As it underestimates the real costs specified in Eq. (6.1), this is an admissible heuristic.

6.1.2 Roadmap Deformation Costs

The key question we left open so far is how to determine the costs of deformations and how to compute the term $C_{def}(i, j)$ for an arbitrary edge (i, j) in the roadmap. In Section 5.1, we discussed a measure for the deformation costs of a robot trajectory that can be obtained by executing corresponding physical simulations. We already ruled out the possibility of performing the corresponding simulations when computing a motion plan as this is too time-consuming for real-world applications. Instead, we make use of our GP-based regression approach to model deformation cost functions for individual objects presented in Section 5.3. Deformation cost functions are defined for robot trajectories relative to an object. In this context, trajectories describe straight line motions between a starting point and end point on a bounding sphere around the object. It is parametrized by a vector $\mathbf{x} = [\theta^s, \phi^s, \theta^e, \phi^e, l]^T$ containing the spherical coordinates of starting point s and end point e in addition to the length l of the motion. In the following, we outline how to determine the deformation costs for an edge in the roadmap with our GP-based deformation cost functions.



(a) Intersections with bounding sphere. (b) Length of motion (1). (c) Length of motion (2).

Figure 6.3: Determining the trajectory parameters of a roadmap edge for GP regression: (a) we compute the spherical coordinates of starting point and end point from intersections with the bounding sphere. (b) The deformation cost function assumes the motion to begin on the bounding sphere, thus the length of the motion is given as l_1. (c) Since the actual starting point of the edge lies inside the bounding sphere, we consider a second trajectory with length l_2. We obtain the final deformation costs of the edge by subtracting the costs for the second trajectory.

To apply GP-regression to an edge in the roadmap, we have to identify its test input vector for the GP, which means computing the corresponding trajectory parameters with respect to an object. We begin with the assumptions that nodes $i, j \in \mathbb{R}^3$ are given in their workspace coordinates and that the trajectory between them is approximately linear. To obtain the workspace coordinates for a given robot configuration, we use the robot's center of mass or the positions of its body parts. As for the second assumption, our roadmap planner connects two nodes by a straight line motion, which is in configuration space, however. For a sufficiently dense sampling of nodes in the roadmap, we assume that the corresponding trajectory in workspace is also approximately linear.

To determine the trajectory parameters, we first compute the positions of starting point i and end point j relative to the object, which can be done with the known transformation of the object in the world model, we denote them by (i_o, j_o). Next, we compute the intersection points i_b, j_b of the trajectory with the bounding sphere of the object and from these compute the spherical coordinates of the trajectory (θ^i, ϕ^i) and (θ^j, ϕ^j). Figure 6.3a illustrates this idea. Finally, we determine the length of the motion, which is given by $l_1 = |j_o - i_b|$, (see Figure 6.3b). This results in the trajectory parameter vector $\mathbf{x}_*^{ij} = (\theta^i, \phi^i, \theta^j, \phi^j, l_1)$ that can be used to evaluate the GP model for an object. The deformation costs returned

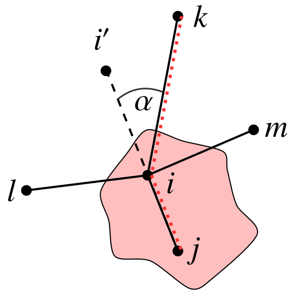


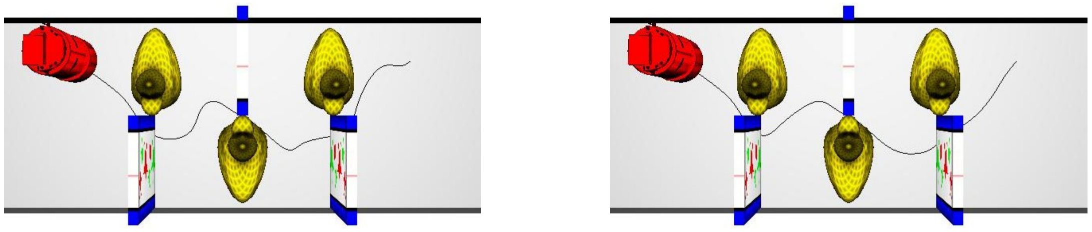
Figure 6.4: When determining the deformation costs for path segment (k, i, j) (indicated in red), we introduce an approximation error: the actual direction from which the robot arrives at node i deviates by an angle α from the direction that is implicitly assumed when computing the deformation costs for edge (i, j). The error gets even larger when considering path segments via nodes l or m.

by GP regression are based on the assumption that the starting point is outside the object. This means the robot is assumed to start its movement not in contact with the object. As Figure 6.3c illustrates, the starting point of an edge can be located arbitrarily inside the bounding sphere of an object, which means the object is already partially deformed, when arriving at this starting point. As this is accounted for when computing the deformation costs of preceding edges, we need to subtract the deformation costs for the trajectory leading to the actual starting point of the edge. We determine the length of this motion as $l_2 = |i_o - i_b|$, and obtain a second GP input vector $\mathbf{x}^{ij}_{**} = (\theta^i, \phi^i, \theta^j, \phi^j, l_2)$. The costs for deforming an object O along an edge then compute as

$$C_{def}(i, j) = \mathrm{GP}_O(\mathbf{x}^{ij}_*, \mathbf{X}'(\mathbf{x}^{ij}_*), \mathbf{y}'(\mathbf{x}^{ij}_*)) - \mathrm{GP}_O(\mathbf{x}^{ij}_{**}, \mathbf{X}'(\mathbf{x}^{ij}_{**}), \mathbf{y}'(\mathbf{x}^{ij}_{**})), \tag{6.2}$$

with $\mathrm{GP}_O(\mathbf{x}^{ij}_*, \mathbf{X}'(\cdot), \mathbf{y}'(\cdot))$ representing an evaluation of the GP model of object O considering the $\mathbf{X}'(\cdot)$ nearest neighboring trajectories and their corresponding outputs $\mathbf{y}'(\cdot)$. To account for the fact that the robot might deform different objects when moving along an edge, we evaluate the GP models of all objects in the environment and sum over the resulting deformation costs. We only need to evaluate a GP model if an edge actually intersects the bounding sphere of an object. Fast line-sphere intersection tests can be used to discard edges with zero deformation costs to keep the number of GP evaluations at a minimum.

We compute the total deformation costs of a path by summing over the deformation costs of subsequent edges. In this way, we introduce an error with our proposed approximation. This error is illustrated in Figure 6.4. When computing the deformation costs of an edge $C_{def}(i, j)$,



(a) Using GPs. (b) Using Simulations.

Figure 6.5: Comparison of our planning system (a) with a planning system that performs simulations during runtime (b).

the GP-regression implicitly assumes that the robot is heading straight to the starting point i, however, the actual direction of the movement leading up to i is given by the preceding edge (k, i) and might deviate by an angle α. The resulting deformation of the object and the corresponding deformation costs when arriving at node i might therefore be different, depending on whether the robot approaches this node via node k, l, or m.

This error could, for instance, be eliminated by rejecting nodes inside the bounding sphere of objects. However, we decided to include samples inside bounding spheres, since it allows the planner to find smoother robot motions. Another possibility to account for this error would be to consider more complex motions when learning the GP models. This, however, would require higher-dimensional inputs and thus significantly larger training data sets. Therefore, we have not investigated this possibility in more detail yet. In our experiments, we investigate the effects of the error introduced by our approximation.

Simulation Example

We compared our planner that determines the deformation costs of path segments using GP regression to a planner that carries out the required simulations during runtime. In a simulation example, the task was to navigate an environment with rubber ducks and curtains, in which deformations of the rubber ducks are more expensive than deformations of curtains. The paths computed by both planners are illustrated in Figure 6.5. Both planners generally avoid the rubber ducks. Our planner underestimates the actual deformation costs of the trajectory by 14 % and the path length deviates by 9.5 % from the optimal solution found by the planner that uses simulations. Our planner, however, is able to compute the plan in less than a second, while the planner that performs accurate simulations requires more than one hour to answer the path query.

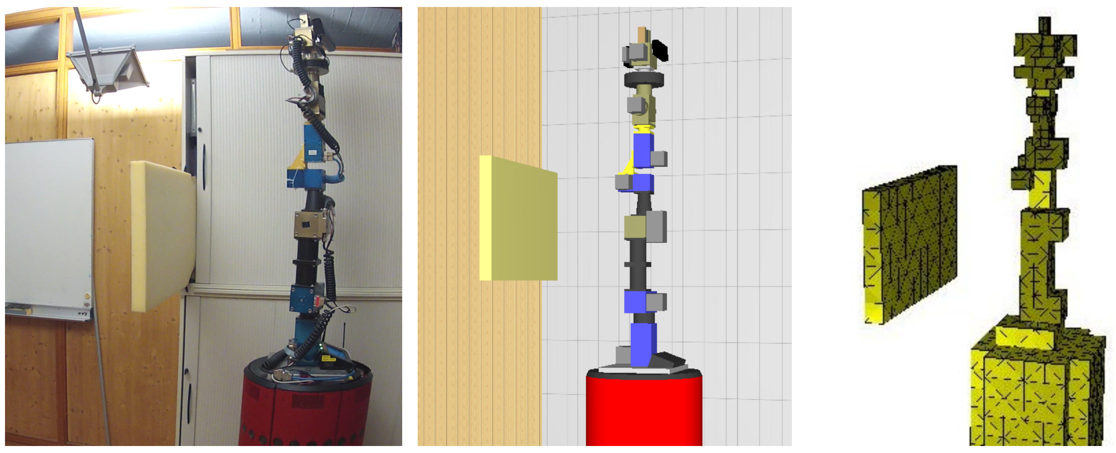

(a) (b) (c)

Figure 6.6: Experimental setup for a real-world planning task: (a) the manipulation robot Zora and a deformable foam mat in its workspace, (b) a 3D model used for roadmap generation and planning, and (c) a deformation model of the foam mat.

6.2 Planning for Manipulators in 3D

In our first application scenario, we implemented a motion planner for our manipulation robot Zora that we already introduced in Section 4 as data acquisition platform for deformation models. In our planning environment, we assume the mobile platform to be stationary and consider only motions of the manipulator. Figure 6.6 illustrates an experimental setup for a planning task, in which we mounted a deformable foam mat in front of the robot acting as a sort of barrier. We implemented the planner within the ZORA framework,[1] which provides functionality for operating the manipulation robot, including hardware drivers, forward and inverse kinematic control, a simulator (without deformation simulation), 3D world modeling, and collision checking.

The manipulator consists of seven rotational joints, accordingly the roadmap represents this seven-dimensional configuration space of the robot within the limits of the joints. The roadmap is constructed in a preprocessing step by uniformly sampling a user-defined number of nodes in configuration space. In our experiments, we set the number of samples to 1000, which resulted in a sufficiently dense coverage of C_{free} for our motion planning tasks. When connecting the generated nodes, we consider for each node its 50 nearest neighbors

[1] J. Sturm: ZORA Software Package. `http://www.informatik.uni-freiburg.de/~sturm/zora-main.html`, last accessed January 7, 2013.

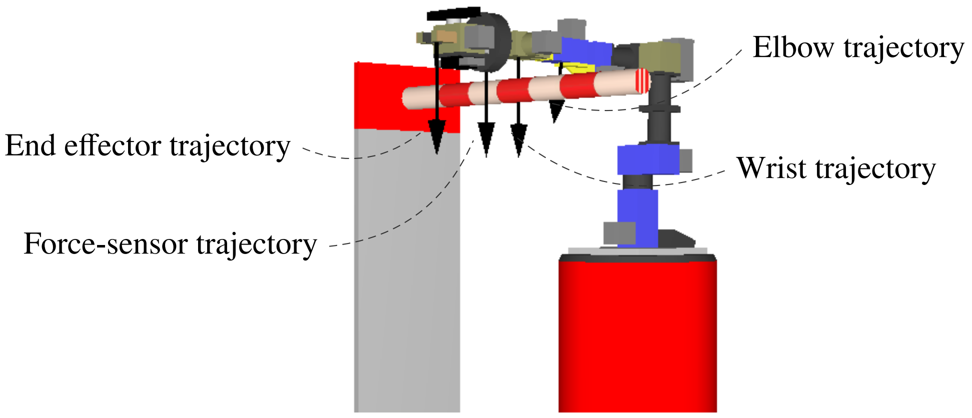


Figure 6.7: Determining the deformation costs of a manipulator motion: the manipulator moves downward in the presence of the deformable bar. We consider the (approximately linear) trajectories described by its individual body parts and compute their deformation costs using GP regression.

and add an edge if the straight-line path in configuration space between both nodes does not lead to collisions with rigid obstacles. This is verified by performing collision checks for 50 intermediate configurations between two nodes. As described above, after connecting initial and goal configuration to the roadmap, we can apply A* to search for a path to the goal that optimizes the trade-off between travel costs in configuration space and deformation costs.

In Section 6.1.2, we explained how to compute the deformation costs of edges in the roadmap that describe straight-line motions of a robot with one degree of freedom in 3D workspace. Now, we discuss how to apply this concept to our manipulation robot that is assembled of different body parts linked in a kinematic chain.

The edges in the roadmap describe motions of the manipulator in configuration space. From these, we determine the corresponding trajectories in 3D workspace that can be used as input for the GP-deformation cost models. The key idea is to describe the motion of the manipulator as a whole in terms of the movements of its individual body parts. Figure 6.7 illustrates this idea. In this example, we consider a simple downward movement of the manipulator in the presence of a deformable bar. The trajectory described by the end effector in this case would not lead to a deformation of the bar and result in zero deformation costs. The main deformation is caused by the movement of the wrist as can be seen in a visualization of the corresponding deformations in Figure 6.8. Thus, to account for deformations introduced by the movement of the manipulator along an edge (i, j) in the roadmap, we sample multiple points along the kinematic chain of the robot. Then, we perform the estimation of the

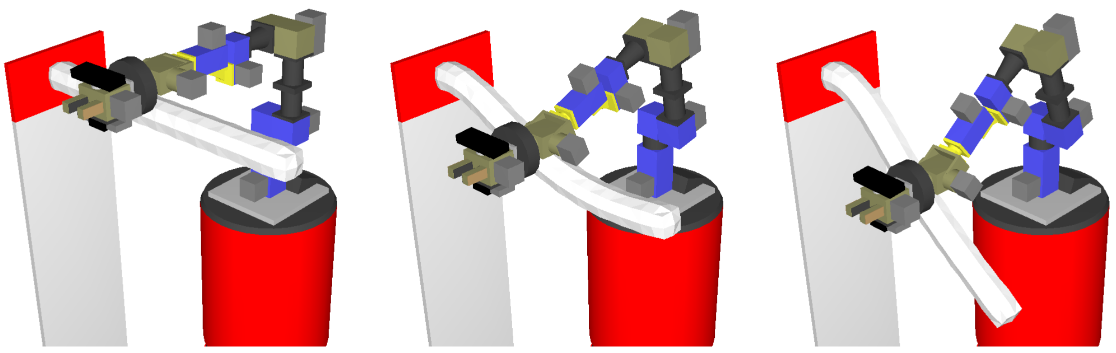


Figure 6.8: Deformation of the bar resulting from the manipulator motion in Figure 6.7.

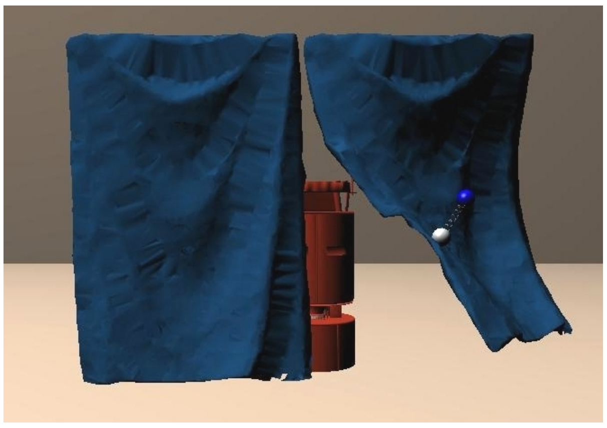
deformation costs for the trajectories of all sampled points along the kinematic chain and consider the maximum of the deformation costs of the individual trajectories

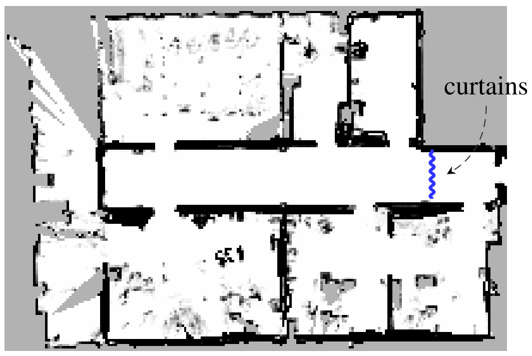
$$C_{def}(i, j) = \max_{b_{ij}} \mathrm{GP}(b_{ij}), \tag{6.3}$$

where b_{ij} refers to the trajectory in 3D workspace described by a body part along an edge (i, j) and $\mathrm{GP}(b_{ij})$ to a GP-Evaluation of the deformation costs of the trajectory as stated in Eq. (6.2). Considering the maximum instead of, for example, the sum generates more accurate predictions since the largest deformation forces typically are generated by one body part only. In theory, there may be situations in which this assumption is not valid, for example, when a robot with two manipulators would squeeze an object – such situations are not considered here.

When determining the deformation costs of roadmap edges, we consider the trajectories described by the individual body parts of the manipulator to be independent instead of considering its full configuration. This approximation allows us to model the cost functions with a low number of inputs. By considering only starting and end point of a trajectory relative to an object, the cost function is furthermore independent of the object's actual location in the environment. In contrast, if we considered the full configuration of the manipulator, the deformation cost function would depend on the position of the object relative to the robot. Furthermore, a higher-dimensional input space to the regression problem would require significantly larger training data sets. In the experiments, we demonstrate that the proposed approach is able to plan reasonable motions leading to minor object deformations and at the same time maintains online performance.

(a)


(b)



(c)

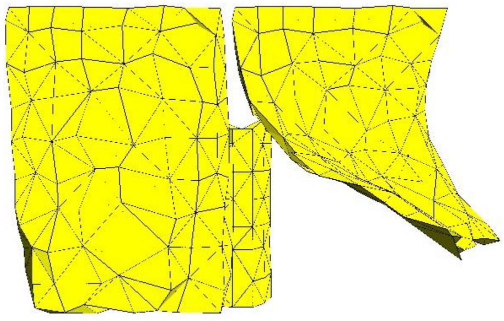

(d)

Figure 6.9: Robot navigation in 2D: (a) our robot Albert in a corridor with curtains, (b) the corresponding deformation simulation, (c) the grid map representing the static part of the world, and (d) the "volumetric" deformation model underlying (b).

6.3 Robot Navigation in 2D

In addition to the manipulation robot scenario, we consider the problem of autonomous navigation of wheeled robots in the presence of deformable objects. We implemented a navigation system for an iRobot B21r platform. The robot we used in our experiments is called Albert and has been an interactive museum tour-guide at the height of its career. The mobile platform, however, is the same as the one of our manipulation robot. Our application scenario is an indoor office environment with a set of deformable curtains mounted in the corridor (Figure 6.9a). Interesting problems arise in this context, when robots navigate in real-world environments: besides motion planning within a model of the environment, a safe and reliable execution of the planned path, including aspects such as localization and collision avoidance, is of major importance. Typically, the robot is equipped with sensors to perceive its surroundings, for instance, ultrasound or laser sensors; when following a path, the robot can use the sensor measurements both for localizing itself within the environment model and for avoiding collisions with unexpected obstacles.

The implementation of our real-world navigation system is based on CARMEN,[2] a robot sensing and navigation software framework allowing independent modules to communicate via middle-ware. It provides modules for low-level robot control and sensing, and furthermore modules for collision avoidance, localization, path planning, and occupancy grid mapping. To integrate our approach into CARMEN, several extensions were necessary.

A prerequisite for autonomous navigation is an appropriate model of the environment. We use occupancy grid maps to represent static obstacles and augment them with information on the position of deformable objects in the environment (see Figure 6.9c). Such a grid map can be learned efficiently from laser range data with the approach of Grisetti et al. (2007) and is used in CARMEN to keep track of the robot's position with the Monte Carlo localization approach of Thrun et al. (2001). To incorporate this model into our planner, we interpret the grid map as a deterministic roadmap with nodes at the centers of the free cells and 4-connectivity between neighboring free cells forming the edges. Estimation of the deformation costs in the roadmap and path planning then can be implemented as described in Section 6.1.2.

The deformation model of the curtain (see Figure 6.9b) deserves a few words of explanation. Cloth really is two-dimensional only, and the elastic deformation model for isotropic material assumed throughout this thesis does not exactly apply, as cloth has low resistance to bending but high resistance to extensions and shear forces. Other deformation models, such as the geometric model proposed by Stumpp et al. (2008) might be better suited to model the characteristics of the curtain. However, to stay within our simulation framework introduced in Chapter 3, we decided on an elastic model: we acquired a geometric model of the curtain with a laser scanner, and from that we generated a (very thin) volumetric representation (see Figure 6.9d) as explained in Section 4.1.2. Furthermore, we manually determined elasticity parameters to achieve realistic deformation simulations. This allows us to compute a deformation cost function for the curtain as described in Chapter 5.

[2]M. Montemerlo, N. Roy, S. Thrun, D. Haehnel, C. Stachniss, J. Glover: CARMEN – Carnegie Mellon Robot Navigation Toolkit. `http://carmen.sourceforge.net/home.html`, last accessed November 23, 2012.

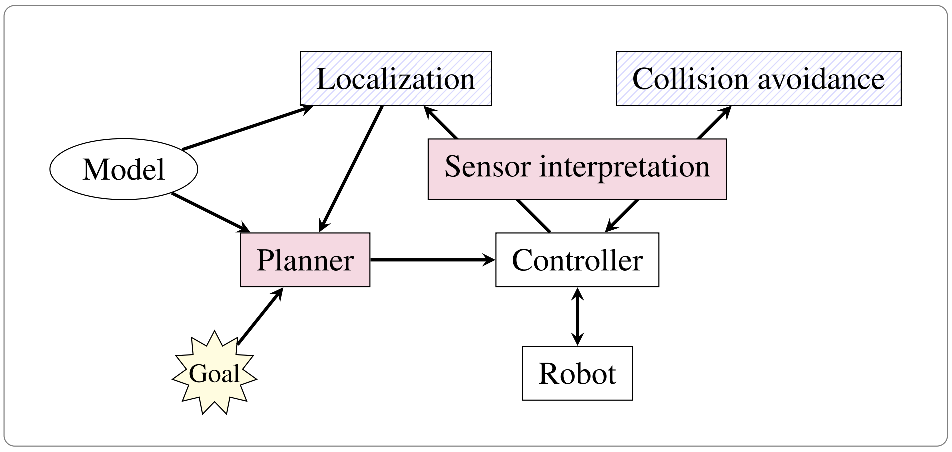


Figure 6.10: The layered CARMEN control architecture for robot navigation: high-level control for planning and localization, low-level control for path following and collision avoidance. We introduced new components to account for object deformations during plan generation and path execution (red). Furthermore, we adapted the collision avoidance behavior and the localization module (blue).

The robot control architecture in CARMEN is illustrated in Figure 6.10. To enable robot navigation in the presence of deformable objects, we replaced the existing planning module with our planner that takes into account the costs of object deformations. In addition to that, we modified the localization module, which is based on Monte Carlo localization, such that laser beams observing a deformable object during deformation are not considered. This is necessary, as the robot is localized with respect to the grid map. The grid map, however, cannot represent deformable objects, in particular their shape changes during deformation. We further modified the collision avoidance behavior of the robot, such that it is able to come in contact with deformable objects. To achieve that, we introduced a new component that interprets the robot's sensor measurements during navigation. In the next section, we address the problem of interpreting the robot's sensor data for localization and collision avoidance.

6.4 Sensor-based Collision Avoidance for Non-deformable Objects

Reactive collision avoidance plays an important role, for instance, when autonomous robots share their environment with humans. Since humans are *dynamic* obstacles, they are not represented in the static model of the environment, and thus, they cannot be accounted for when planning the robot's motions. Approaches to reactive collision avoidance, in general, prevent the robot from getting too close to any obstacle. In our scenario, however, the robot might be required to deform an object while following a path, which involves contact with it. Consider, for instance, the trajectory that guides the robot through the curtains in Figure 6.9a. In this example, the robot is in close contact with the curtain, and its laser sensor is partially occluded by the object, thus reporting short distance measurements. In such a situation, no collision avoidance behavior is required. But imagine a person appearing behind the curtain. In this case, the robot should definitely be able to avoid the collision. Hence, a new challenge arises, that is how to interpret the sensor data of the robot and how to distinguish measurements corresponding to deformable objects from measurements belonging to rigid and dynamic obstacles that are to be avoided.

Our robot is equipped with a SICK laser range scanner with an opening angle of 180 degrees and an angular resolution of one degree. This scanner is mounted inside the base of the robot at a height of approximately 40 cm, therefore allowing the robot to observe a two-dimensional slice of the world in front of it with 180 individual laser beams at a time. To reason about the deformability of an observed object and to classify the robot's sensor measurements, we combine knowledge about the robot position and objects in the environment with expected laser range scans during deformations. We model this problem in a probabilistic fashion: let c_i denote the binary random variable describing the event that beam i observes a deformable object. Then, we are interested in $p(c_i \mid x, z_i)$, the probability that beam i corresponds to a deformable object given the robot is at position x and measures the range z_i. Applying Bayes' formula, we arrive at

$$p(c_i \mid x, z_i) = \frac{p(z_i \mid x, c_i)p(c_i \mid x)}{p(z_i \mid x, c_i)p(c_i \mid x) + p(z_i \mid x, \neg c_i)p(\neg c_i \mid x)}. \tag{6.4}$$

Here, $p(z_i \mid x, c_i)$ is the sensor model for a deformable object, it models the probability distribution of a sensor measurement that corresponds to a deformable object; $p(c_i \mid x)$ is

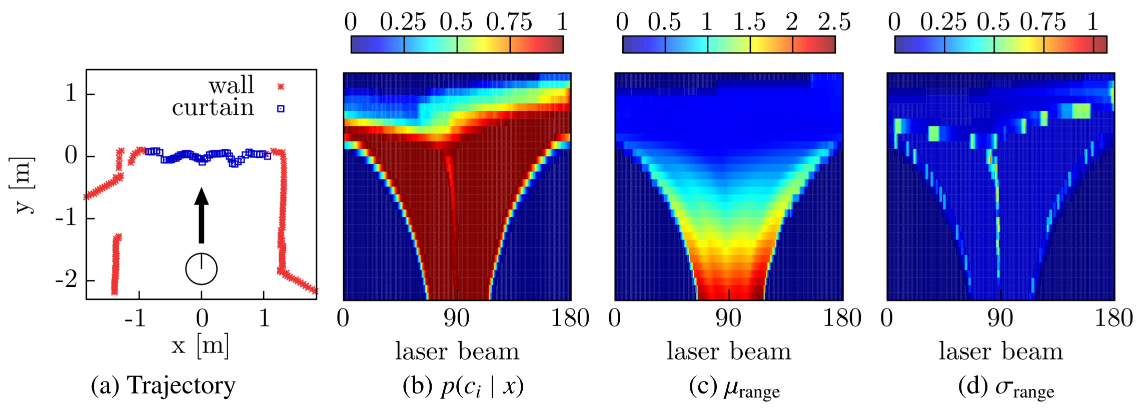


Figure 6.11: Sensor model for observing the curtain along the trajectory (a): the figures show the probabilities $p(c_i|x)$ (b), the average expected beam length (c), and the corresponding standard deviation (d) when observing the deformable object.

the prior denoting the probability of observing a deformable object from position x. We will shortly go into detail of how to learn these models. The sensor model $p(z_i \mid x, \neg c_i)$ corresponds to the sensor model $p(z_i \mid x)$ assessing the probability of a measurement given the robot is at position x. This sensor model, together with the motion model of the robot, is one of the key ingredients to the Markov localization approach that updates the robot's belief about its current position in the environment. It models noise characteristics of the sensor and is described in detail by Fox et al. (1999). For a given static environment, it can be precomputed, and we use the sensor model incorporated into the localization module and provided by the Carmen framework. To compute the probability that an observation belongs to a deformable object in Eq. (6.4), we further require the position of the robot to be known. We work with the pose estimate provided by the localization module.

6.4.1 Learning a Sensor Model for Deformable Objects

The sensor model $p(z_i \mid x, c_i)$ states the probability of an observation z_i given the robot is at position x and the observation in fact belongs to a deformable object. The prior $p(c_i \mid x)$ denotes the probability of observing a deformable object from position x. Effectively, these quantities not only depend on the robot position, but also on the trajectory leading to the position. Obviously, an object is deformed differently, if at all, depending on the direction from which the robot approaches a position, as we already observed when defining the deformation cost function of a robot trajectory in Section 5.1. Accordingly, also the laser sensor will measure different distances depending on the trajectory. To account for that, we determine sensor models that correspond to different trajectories of the robot relative to an object. We restrict ourselves to a set of different straight trajectories that are typically chosen by our path planner.

To determine the sensor model for a trajectory, we record different data sets of the robot deforming the object along a trajectory. Each data set consists of the robot positions x (provided by the localization module) and the measured ranges z_i along a trajectory. We manually label the beams that are reflected by a deformable object. From the labeled measurements obtained along these trajectories, we compute the statistics

$$p(c_i \mid x) = \frac{\text{hit}_{def}}{\text{hit}_{def} + \text{miss}_{def}}, \tag{6.5}$$

where hit_{def} is the number of beams that are reflected by a deformable object and miss_{def} states how often no deformable object was observed for a given position x and viewing angle i. The sensor model $p(z_i \mid x, c_i)$ is described by a Gaussian with average range μ and variance σ^2. We obtain it by computing the mean and variance of the measurements of the training data sets. Both, the sensor model and the prior of observing a deformable obstacle, are represented in a discretized state space of the robot position, with the resolution of the occupancy grid map. An example of such a sensor model for a typical robot trajectory through the curtain is shown in Figure 6.11. This representation is dependent on the robot trajectory but can be considered independent of the absolute location of the object in the environment, as it only accounts for measurements relative to the deformable object, not for measurements that might be influenced by the static part of the environment.

6.4.2 Classifying Sensor Measurements and Avoiding Collisions

When navigating autonomously, the robot has to constantly monitor its sensors to keep track of its movements (e. g., with wheel encoders) and the obstacles in its vicinity (with the laser range-finder). These measurements are incorporated into the localization algorithm to maintain a probability distribution over the robot's current position and into the collision avoidance system to account for unforeseen obstacles. In our application, the robot should only consider measurements that correspond to the static part of the world or to dynamic obstacles. We achieve this by filtering out the range measurements that have a high probability of observing a deformable object. Whenever a new range scan is available, we

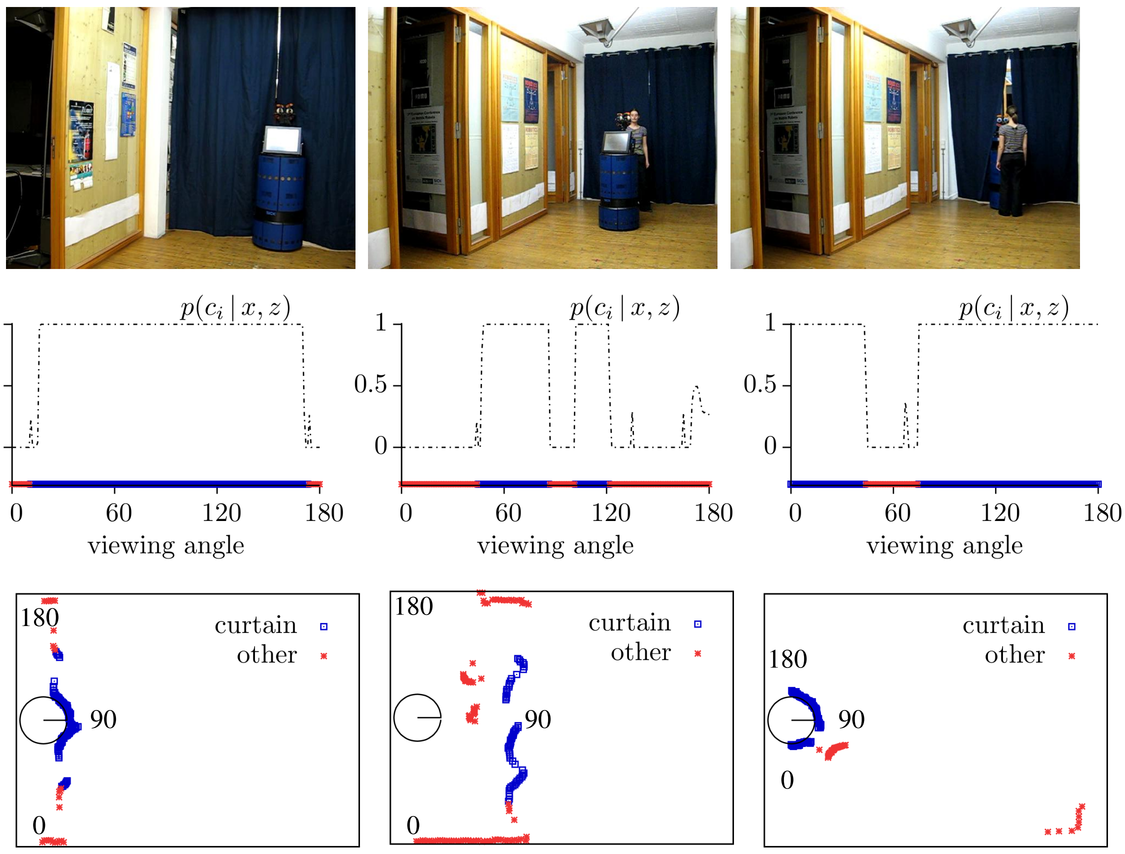


Figure 6.12: Different collision avoidance scenarios (top row): Laser beams are evaluated with respect to their likelihood of observing a deformable object (second row). The bottom row illustrates the classification of the individual laser beams.

evaluate Eq. (6.4) for each range measurement. If this probability exceeds a threshold of 0.5 in our implementation, we classify the measurement as corresponding to a deformable object. This filtering of the range measurements can be used with any collision avoidance technique that can adapt the robot's motions to dynamic obstacles, for instance, the dynamic window approach of Fox et al. (1997) or the nearness diagram technique of Minguez and Montano (2000). In our collision avoidance system, we try to drive the robot around dynamic obstacles by incorporating the measurements, which are identified to belong to dynamic obstacles, into the map and replanning. However, if an obstacle is closer than a minimum distance of 0.5 m, we stop the robot. Figure 6.12 illustrates different collision detection scenarios. In the first example, the robot is in close contact with the curtain, and, as it is able to correctly classify the measurements, it continues its trajectory. In the second and third situation, a person's legs are observed close to the curtain. The system is able to distinguish them from the curtain and therefore stops the robot to avoid a collision.

6.5 Experimental Results

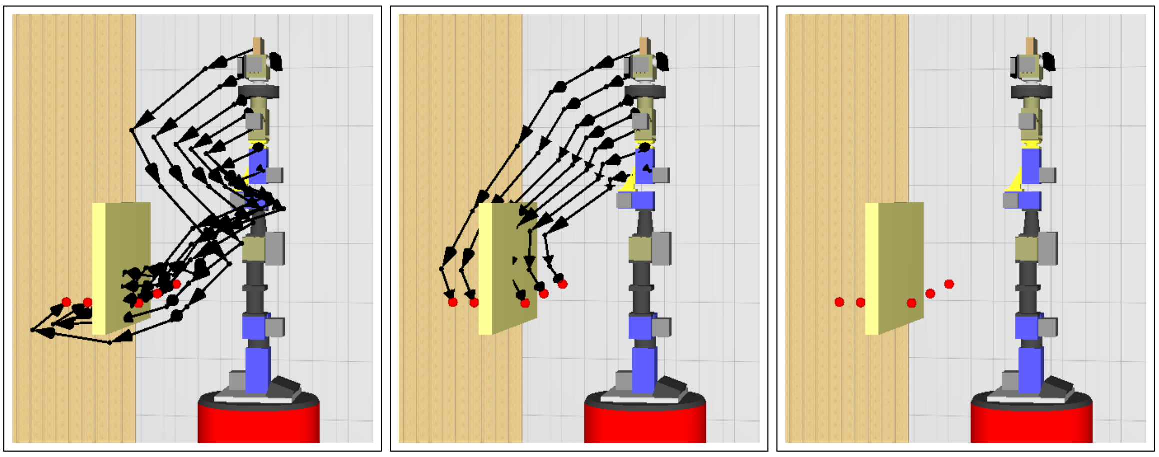
In this section, we present example applications and experimental evaluations of our proposed planning system. We first demonstrate the planning system for our manipulator. Second, we investigate the navigation scenario with the wheeled robot in more detail. For this application, we evaluate the planning algorithm as well as the collision avoidance.

6.5.1 Arm Planning in 3D

We set up an experimental environment with a deformable foam mat for our manipulation robot Zora (shown in Figure 6.6). We generated a roadmap accounting for the static part of the world. The roadmap contains 1,000 configuration samples and 8,635 connections between nodes. Precomputing the roadmap for the static part of the world required approximately 30 m, including sampling configurations, connecting nodes and performing the corresponding collision detections. We evaluated the deformation costs of edges using our local GP-regression approach introduced in Section 5.3 with a set of 22,950 precomputed trajectory samples, which were already used for evaluation purposes in Section 5.5. The local GPs were built using the neural network covariance function, and the hyperparameters were optimized once for a subset of 2,000 trajectories. The deformation costs of edges can be evaluated for the roadmap before answering any queries using our approximation described in Section 6.2. For 6,358 edges, this required another 420 s. With the roadmap precomputed in this way, arbitrary goal positions can be queried, only new edges connecting the initial and goal configuration need to be evaluated using GP regression at query time.

Path Queries

As an example planning task, the manipulator was required to move from its initial position, in which the arm is stretched upwards to a goal configuration facing forward, in which the goal position of the end effector is behind a deformable foam mat. The path generated by our planner is visualized in Figure 6.13a, it shows the workspace trajectories of different manipulator body parts along the edges of the roadmap. To minimize the deformation of the foam mat, the planner chooses a motion that approaches the target position from the front and slightly below. The movement of the manipulator along this path is shown in Figure 6.14.



(a) Considering deformation costs. (b) Ignoring deformation costs. (c) Considering the object as rigid.

Figure 6.13: Different motion plans to reach the goal configuration behind the foam mat. (a) Our planner minimizes the trade-off between motion and deformation costs. (b) A planner ignoring deformable obstacles chooses the shortest path. (c) A planner treating all obstacles as rigid is not able to find a path to the goal.

To illustrate the advantage of considering object deformations when planning motions, we compare our planner to two alternative planners, one that ignores deformable obstacles, and one that treats them as rigid obstacles. The plan generated when completely ignoring the foam mat is shown in Figure 6.13b. Executing this plan results in tearing down the foam mat, as can be seen in snapshots of the robot movement in Figure 6.15. Our deformation model assumes the object to be fixed, it does not account for tearing down the object, but still it is able to model the high deformation costs of such motions. In comparison to the planner ignoring the object, our planner introduces a computational overhead of 2.4 s per query. In total, 8.6 s were required to compute the path. A planner treating all obstacles as rigid is not able to find a path, since the goal configuration leads to a collision with the foam mat. This fact is illustrated in Figure 6.13c. The successful execution of the path from Figure 6.13a is furthermore illustrated in a video that can be found online.[3]

[3]Real-world experiment: `http://www.informatik.uni-freiburg.de/~bfrank/videos/zora_foam.avi`

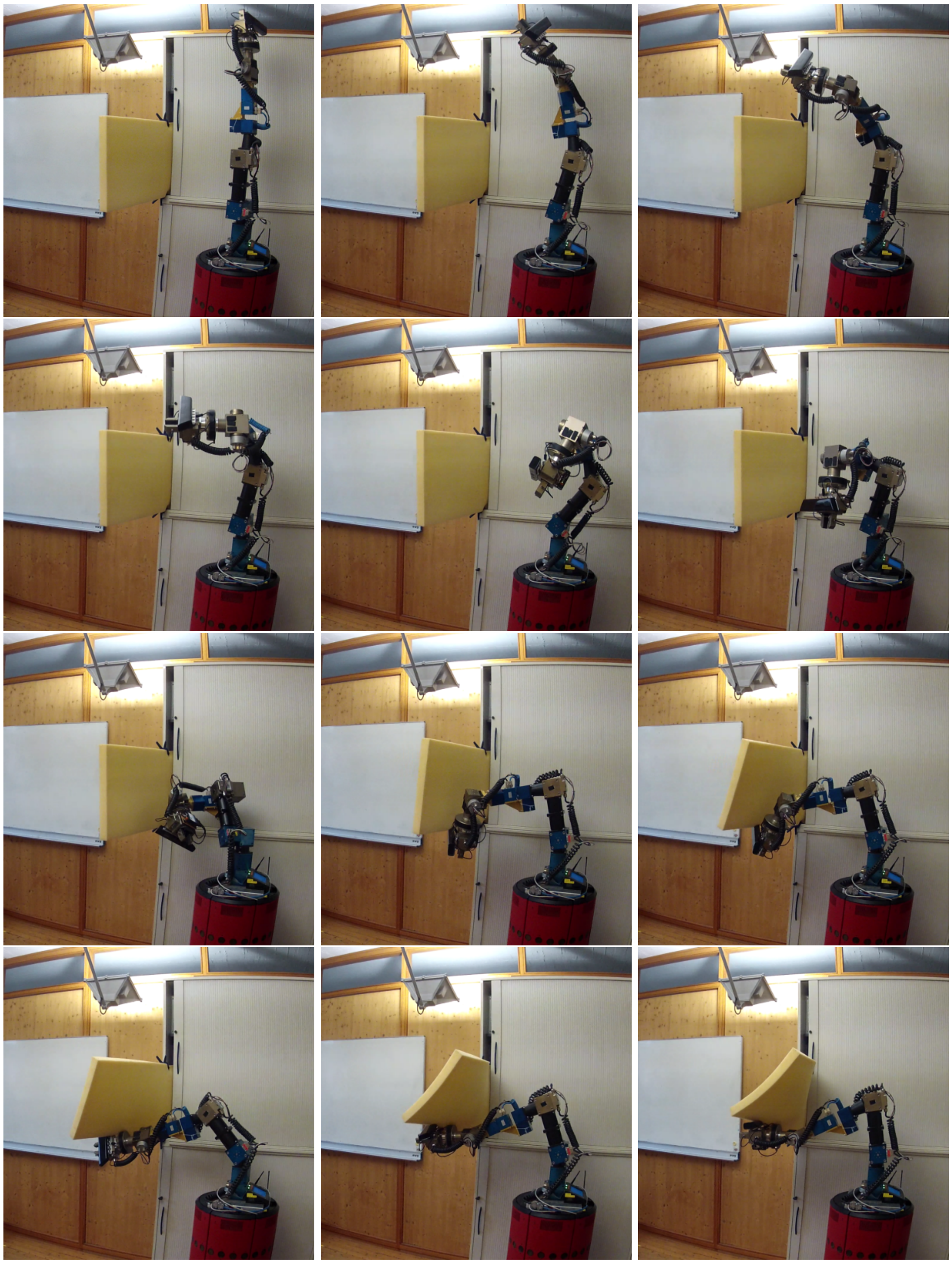


Figure 6.14: Our robot Zora moves along the planned trajectory from Figure 6.13a, keeping the deformation of the foam to a minimum.

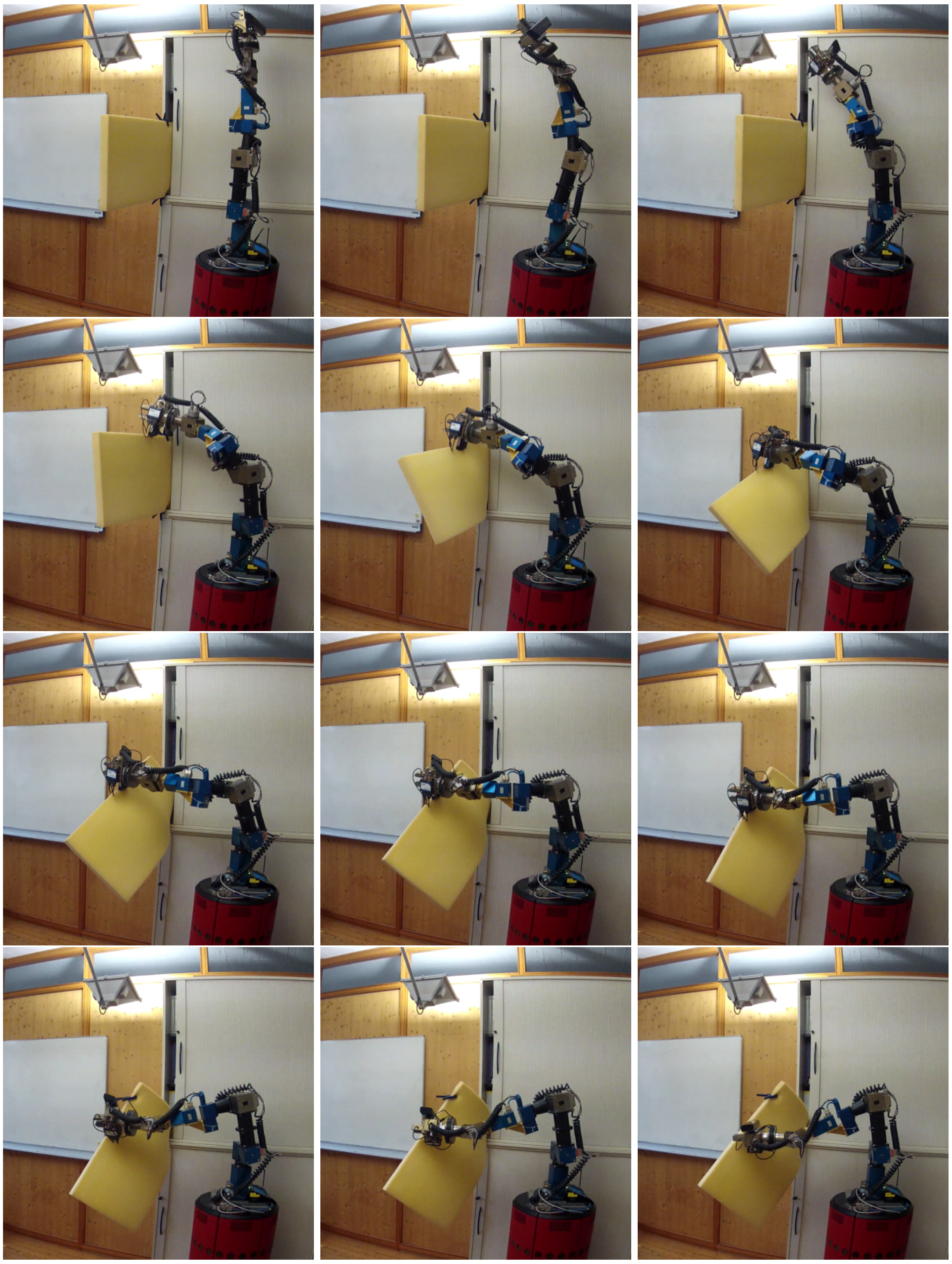


Figure 6.15: When executing the shortest path that ignores deformable objects (Figure 6.13b), Zora tears down the foam mat and destroys the experimental setup.

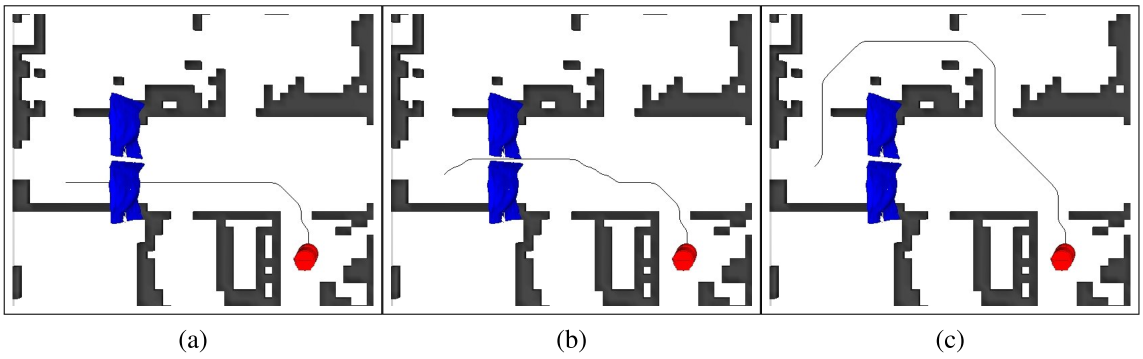


Figure 6.16: Planning a path for different weightings of the deformation costs: with $\alpha = 0$, the deformation costs are ignored (a), for $\alpha = 0.2$, a longer trajectory is chosen to minimize deformations (b), $\alpha = 0.8$ leads to avoiding deformations (c).

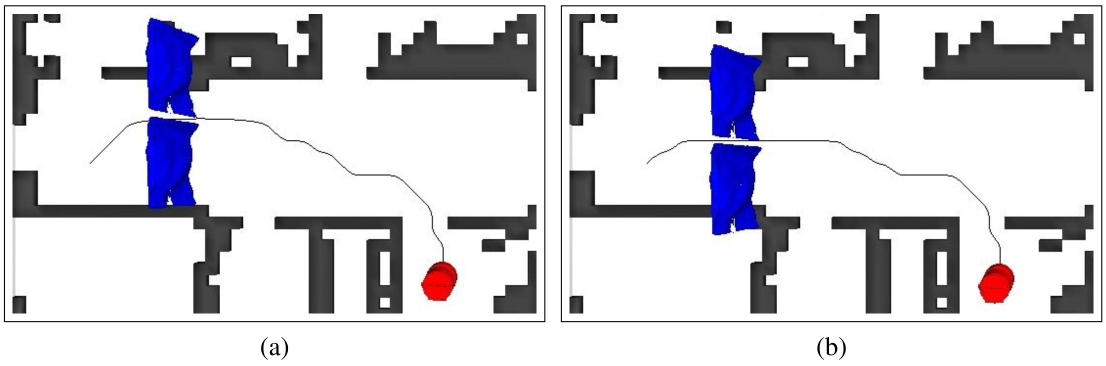


Figure 6.17: The planner prefers trajectories that minimize object deformations. The curtains in setup (a) are moved 40 cm along the positive y-axis compared to the setup from the previous experiment (b). The weighting coefficient α is set to 0.2 in both examples.

6.5.2 Robot Navigation in 2D

We evaluated our navigation system described in Section 6.3 on our robot Albert, a wheeled platform equipped with a laser range scanner. To set up a navigation scenario, we mounted two curtains in the corridor of our lab as deformable objects. The robot in this environment is shown in Figure 6.9. We performed several experiments to evaluate our motion planner as well as our approach to classify the sensor measurements of the robot during navigation.

Path planning

In the environment with the deformable curtains, the robot is faced with the task of reaching a goal point beyond the curtains. The planner optimizes the weighted sum of travel costs and deformation costs when searching for a path, and a weighting coefficient α (see Eq. (6.1)) determines their trade-off. In a first experiment, we investigated the influence of this weighting coefficient on the generated trajectories. For fixed starting points and goal points, we varied the weighting coefficient α and compared the trajectories generated by our planner. The results for an example planning task can be seen in Figure 6.16. In our setup, the deformations of the curtains are minimized if the robot moves on a trajectory between both curtains and thereby deforms them equally at their borders only. We found that for low values of $\alpha \approx 0.2$, the planner prefers trajectories with low total costs, it avoids large detours and minimizes object deformations. This fact is illustrated in a second experiment, in which we varied the experimental setup and moved both curtains. Figure 6.17 shows the generated trajectory for this setup and compares it to the trajectory determined for the previous setup. In both cases, α is set to 0.2. The planner chooses a somewhat longer trajectory in order to minimize the deformation costs.

Sensor interpretation

In the next experiment, we evaluated how well our sensor model for deformable objects is able to predict the presence of deformable objects during robot navigation. We learned sensor models for two different trajectories through both curtains that were chosen preferably by our planner. These trajectories correspond to the two possible trajectories from one side of the curtains to the other with minimal deformation costs. To compute the sensor model statistics for each trajectory, we recorded twelve data sets consisting of laser data and robot positions along the trajectories. We manually labeled the laser beams that were reflected by the curtain. For each trajectory, we performed a leave-one-out cross-validation using eleven data sets for learning the model and one for evaluation. The results of this experiment are summarized in a confusion matrix in Table 6.1 and demonstrate that the system is able to distinguish between deformable and static obstacles with high accuracy. While the number of false positives is at around 3 %, the number of false negatives is below 1 %.

		True class	
		Deformable	Non-deformable
Predicted class	Deformable	43857 (97.1%)	621 (0.9%)
	Non-deformable	1292 (2.9%)	65907 (99.1%)
	Total	45149	66528

Table 6.1: Confusion matrix for predicting, whether a sensor measurement corresponds to a deformable object in a static environment.

		True class	
		Deformable	Dynamic
Predicted class	Deformable	8563 (96.5%)	98 (2.1%)
	Dynamic	314 (3.5%)	4600 (97.9%)
	Total	8877	4698

Table 6.2: Confusion matrix for predicting, whether a sensor measurement corresponds to a deformable object in an environment containing both deformable and dynamic objects.

Recognition of dynamic obstacles

In a further experiment, we evaluated the classification of sensor measurements in the presence of dynamic obstacles. Intuitively, the sensor model is able to distinguish well between deformable and static non-deformable objects contained in the map of the robot. The key question, however, is whether the system is able to distinguish well between deformable objects and close-by dynamic obstacles, provided that the dynamic obstacles are not occluded by deformable objects and can be perceived by the sensor. To evaluate how well the classification works, we performed several experiments, in which the robot moved on a trajectory deforming the curtain while a human was blocking its path. The recorded laser scans were labeled accordingly and evaluated with respect to the prediction performance. The results are listed in Table 6.2. In this experiment, the number of false negatives is comparable to the situation in static environments while the number of false positives is around 1 % higher than in the previous experiment. Our experiments, however, showed that this still leads to a safe navigation behavior. In the worst case, false negatives forced the robot to unnecessarily stop while the false positives usually were outliers in a region of correctly classified measurements observing a dynamic obstacle. Therefore, the robot was still able to recognize dynamic obstacles and thus able to avoid collisions with these obstacles.

Real-world navigation example

In a navigation example task, we demonstrate the capability of our system to integrate path planning and collision avoidance and to navigate safely in the environment described above. Figure 6.18 shows a sequence of snapshots of our real robot moving through the curtains. A video of the robot navigating in this environment and demonstrating its ability to avoid collisions with dynamic obstacles can be found online.[4]

6.5.3 Computation Time

In this section, we analyze the computational cost of our proposed motion planner. Besides the computationally intense generation of training examples for the GP deformation cost function (5.5 h for the curtain / 24 h for the foam mat, see Section 5.5), a model for the static part of the world has to be determined. The occupancy grid map for the 2D navigation example was determined independently and readily available with the CARMEN software package. The roadmap computation for the static part of the manipulator environment took approximately half an hour.

Given a model of the static part of the environment and a set of trajectory training samples for deformable objects, the deformation costs for movements represented in the static model can be evaluated using our GP-based regression approach. The deformation costs for the 2D grid map can be precomputed in 3 s, for 800 edges that potentially lead to object deformations. To evaluate the deformation costs of 6,358 edges potentially leading to object deformations in the 3D roadmap, 420 s were required.

With these precomputations available, we evaluated the time required to solve path queries in 10 runs with random starting and goal points. In the 2D environment, path queries could be answered in 0.3 s on average. In the 3D manipulator setting, answering one query required on average 8 s, including collision checks for connecting new nodes, evaluating the deformation costs of new edges and searching for a path. Evaluating the deformation costs introduces an overhead of 2.5 s. Thus, path queries can still be solved efficiently, even for manipulation robots with many degrees of freedom.

[4]Real-world experiment: `http://www.informatik.uni-freiburg.de/~bfrank/videos/albert_curtains.avi`

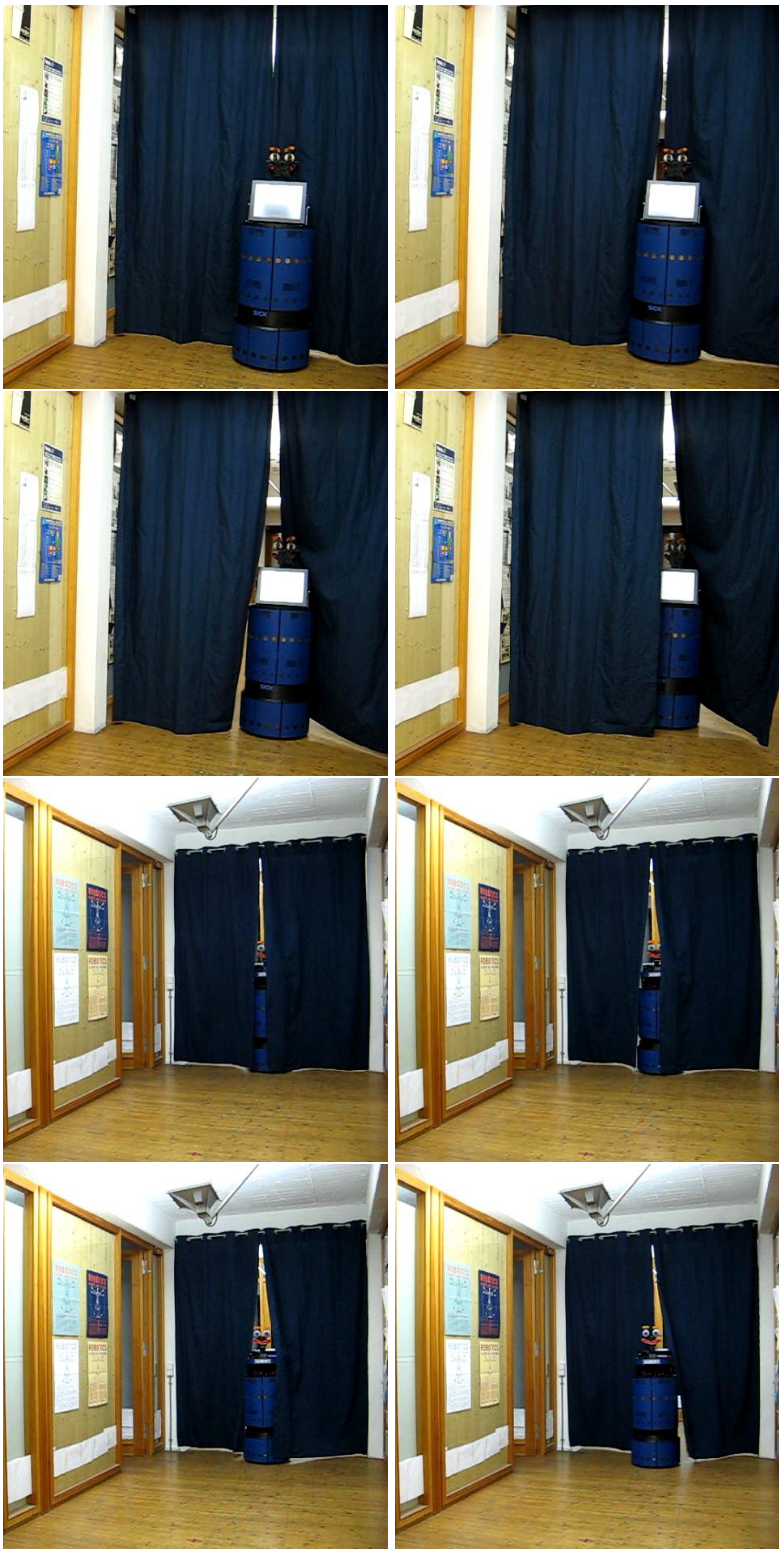


Figure 6.18: The mobile robot Albert is navigating through curtains. The first four snapshots show the robot approaching the curtains and the last four snapshots show the robot passing through from the other side of the corridor.

With a small thought experiment, we further illustrate the efficiency of our proposed planner. Instead of precomputing sample trajectories and estimating the deformation costs of edges in the roadmap using GP regression, it would be possible to perform the simulations of the motions along edges directly when constructing the roadmap. Considering the roadmap from the manipulation robot example and assuming a computation time of 3 s for the simulation of an edge, evaluating 6,358 edges would require an additional 5 h when constructing the roadmap. Furthermore, when answering path queries, the initial and goal configuration would need to be connected to the roadmap. In the worst case, assuming that each node is connected to its 50 nearest neighbors, this would require 200 simulations (two simulation runs are necessary per edge) and another 10 m per path query, thus increasing the runtime for answering path queries by approximately two orders of magnitude. Similar to our approach relying on GP regression, this precomputation represents an approximation to the true deformation costs along an edge in the roadmap. The deformation costs of edges are considered to be independent while in fact, they depend on their preceding edges. To accurately consider the effects of object deformations, the corresponding simulations would have to be carried out when answering path queries. In that case, however, answering one query could easily take hours. In contrast, our planner is able to answer path queries in the order of seconds, and in this way facilitates a prompt response of the robot to new motion tasks.

6.6 Summary

In this chapter, we introduced our motion planning framework that considers object deformations when planning the motions of a robot. It is based on probabilistic roadmaps and optimizes the trade-off between travel costs and object deformation costs. We discussed how to efficiently determine the deformation costs along robot trajectories using the object deformation cost functions introduced in Chapter 5. The proposed object deformation cost functions achieve a speed-up of several orders of magnitude when answering path queries, compared to a planner that performs the required deformation simulations online. This was demonstrated in experiments in 2D and 3D environments

For wheeled robots navigating in deformable environments, we presented an approach to learn sensor models for deformable objects. These models enable a robot to distinguish observations of deformable objects from observations of dynamic obstacles when executing a path. In this way, the collision avoidance behavior can be adapted such that the robot is allowed to come in contact with deformable objects but still reliably avoids collisions with other obstacles, for instance with humans populating the environment.

We implemented the proposed planning system on two different robotic platforms, a manipulation robot with seven degrees of freedom and a wheeled platform. In experiments, we demonstrated that the system can be successfully applied to real robots and enables them to achieve tasks and reach goal points that would be inaccessible if the deformation properties of obstacles were neglected.

7 Related Work

In this thesis, we have addressed two different research areas, on the one hand modeling of deformable objects and estimating their elasticity parameters, and on the other hand planning motions for robots considering these models. We have presented an approach to determine deformation models of real objects and an approach to use these models in a motion planner. In the following, we provide a literature review on related work in both these areas. While modeling of deformable objects receives much attention from the computer graphics community, planning is an area of interest in the robotics community. One application that is unifying these two fields is the simulation of surgical procedures. Computer graphics is concerned with the realistic simulation and visualization of deformable tissues, robotics approaches have been developed that build on these models and employ them to plan the motions of surgical tools. We will first give an overview of the state of the art in deformable modeling and discuss approaches to the estimation of elasticity parameters related to our approach before we turn to the field of robot motion planning and, in particular, planning approaches that consider deformable objects.

7.1 Deformable Modeling and Parameter Estimation

7.1.1 Deformation models

Modeling of deformable objects has been an active area of research for some time now, with applications to games, movies, or surgical simulation. Gibson and Mirtich (1997) provide an overview of deformation models including mass-spring and finite element models. This report is updated by Nealen et al. (2006). In earlier years, mass-spring systems have been frequently used to represent non-rigid objects and to simulate deformations, for instance, in the works of Chadwick et al. (1989), Bridson et al. (2002), or Conti et al. (2003). They are easy to implement and can be simulated efficiently. While mass-spring models are physically motivated and able to handle large deformations, their major drawback is the tedious modeling. As Nealen et al. (2006) pointed out, there is no intuitive relation between spring constants and physical material properties in general. Therefore, determining spring constants that accurately model an object can be difficult. A physically more plausible model that additionally considers constraints for mass-spring systems, such as volume and surface area preservation was proposed by Teschner et al. (2004) and allows for the interactive simulation of thousands of primitives.

With the availability of more powerful computers, finite element methods (FEMs) became increasingly popular. They model the deformation behavior of objects based on continuum elasticity theory and therefore reflect the physical properties of objects in a more natural way. Bathe (1996) and Chandrupatla and Belegundu (2002) provide thorough treatises of FEMs for engineering applications. The seminal work of Terzopoulos et al. (1987) discusses physically based deformation models based on continuum mechanics for simulation. Bro-Nielsen (1998) proposes an efficient linear FEM formulation for surgical simulations that only computes the deformations of the surface nodes. Debunne et al. (2001) present an adaptive formulation for interactive haptic rendering. To deal with the high computational complexity of nonlinear FEMs, Wu et al. (2001) consider adaptive meshes. Linear FEMs provide accurate results only for small strains and introduce ghost forces when applied to large rotational deformations. To avoid nonlinear computations and eliminate these ghost forces, Hauth and Strasser (2004) and Müller and Gross (2004) proposed a computationally efficient corotational finite element approach, which is also used in our simulation framework.

7.1.2 Parameter estimation

In computer animation applications, the physical parameters of models are typically adjusted manually such that they lead to visually plausible results. This, however is not desirable in simulations of real-world behavior. For instance, in surgical simulations, the parameters should be related to the material properties. Also in haptic rendering, the generated forces are required to approximate real material behavior.

There are several different approaches to determine the parameters of mass-spring models. Bianchi et al. (2004) learn the stiffness constants of mass-spring models using a genetic algorithm and comparing it to an finite element reference model. The identification of mass-spring parameters is also discussed in the work of Lloyd et al. (2007). They derive an analytical formulation for the spring parameters from the linear finite element model for different mesh topologies. Another approach that estimates the stiffness properties of mass-spring models was proposed by Burion et al. (2008). They use a particle filter to obtain a posterior distribution over the stiffness parameters and evaluate the particles by comparing simulated and observed deformations. In contrast to our work, these approaches use mass-spring models and furthermore, they work on simulated data only.

The use of data-driven representations for deformable objects was proposed, among others, by Fong (2009) and Bickel et al. (2009). Fong (2009) presents a system to measure deformations of elastic objects using a structured-light camera and a force-sensor. He extracts force-fields for different contact points and displacements on the objects. For haptic rendering of unseen contact points, the forces are interpolated using radial basis functions. In a similar way, Bickel et al. (2009) introduce a data-driven representation of heterogeneous and nonlinear material by fitting radial basis functions to measured force-displacement samples. They, however, use an underlying linear finite element model, similar to our approach, to model the homogeneous parts of objects.

In material science, the so-called tensile test is frequently used to determine physical material parameters of objects (Hart, 1967). In this test, uni-axial loading is applied to a material sample, thereby measuring the stresses and strains, until the material fractures. From these measurements, parameters such as Young's modulus and Poisson's ratio can be determined. Since it requires an accurate testing apparatus, and test objects are destroyed in the process, this test is less suited to our application.

Inverse modeling approaches deduce the parameters of a system using observations of the system behavior (Tarantola, 2005). Many approaches following this general idea have been proposed to determine the elasticity parameters of real objects. This can be done by optimizing an objective function that relates the observations to a finite element simulation, which in turn depends on the parameters to be determined. For instance, Kajberg and Lindkvist (2004) determine the material parameters of thin metal sheets including plasticity effects by minimizing a least-square functional. Choi and Zheng (2005) present an approach to determine Young's modulus and Poisson's ratio of soft tissues from indentation tests. Schnur and Zabaras (1992) determine different parameters, including Young's modulus, of a two-dimensional nonlinear finite element model by solving a nonlinear least squares problem. They assume Poisson's ratio to be given. The approach of Becker and Teschner (2007), in contrast, works for three-dimensional objects, allows for the simultaneous estimation of Young's modulus and Poisson's ratio, and furthermore can be reduced to a linear least squares problem. Both approaches, however, have been validated using simulated data only. Zantout and Zheng (1994) present a method to extract the stiffness parameters of thin plates based on a 2D finite element method and measure displacements using geodesics. They assume deformations to be generalized bending but present no verification on real data.

Estimation of material parameters operating on real data has been investigated in the context of soft-tissue modeling for surgical simulation applications, such as simulation and training, or computer-aided surgery. Kauer et al. (2002) present an inverse finite element algorithm that estimates the material parameters of soft biological tissues. They consider complex material constitutive laws, such as nonlinearity and anisotropy, furthermore they account for viscoelastic behavior. They measure deformation forces of soft organic tissue with an aspiration instrument that generates a weak vacuum and measures the pressure, and at the same time observe the deformed surface with a camera. Their estimation procedure is designed to operate on two-dimensional image data only. Another approach that collects data of real objects was presented by Lang et al. (2002). They describe a deformation model as a discrete boundary value problem and estimate Greens' functions from measured forces and displacements by formulating this as a linear optimization problem. They collect data of object deformations with a complex robotic measurement facility, including force sensors and different stereo cameras. Fugl et al. (2012) present an approach to determine Young's modulus and different parameters to model heterogeneous material from observations of deformations that are due to gravity with an RGB-D camera. An interesting approach was recently presented by Boonvisut et al. (2012). They collect tissue deformation data and force data with a robotic manipulator that performs a deformation trajectory. In this approach, different deformation trajectories can be considered, and the parameters are optimized such that a finite element simulation of the trajectory agrees with the observed trajectory.

Our approach also belongs to the class of inverse finite element parameter estimation procedures. In contrast to most of the previous approaches, our method deals with real data and has been realized on a real mobile manipulation robot that can actively deform objects. In our setup, the robot furthermore carries its sensors on-board and thus is the basis for fully autonomous exploration. So far, manual interaction is still required to acquire deformation data of objects, but an envisioned application of our approach could be an indoor autonomous household robot, that is equipped to explore its environment and learn about deformable objects contained therein.

7.2 Robot Motion Planning and Learning

7.2.1 Robot Motion Planning

Robot motion planning has been studied for decades now as one of the fundamental problems towards autonomous agents. Different planners suited to all kinds of robots, environments, and situations have been developed, mostly assuming the environment to be static and obstacles as well as robots to be rigid. The textbooks of Latombe (1991) and, more recently, Choset et al. (2005) andLaValle (2006) provide thorough treatises of the subject. For wheeled platforms, efficient planners operating on occupancy grid maps have been developed, for instance, the dynamic A* algorithm (D*) of Stentz (1994), and variants thereof introduced by Ferguson and Stentz (2006) or Likhachev et al. (2005). They address the incorporation of new sensor information in partially known or dynamic environments and are thus able to efficiently repair or refine initial plans.

Sampling-based approaches have been introduced to address planning for complex systems with many degrees of freedom. The rapidly exploring random trees (RRTs) were originally developed for motion planning under differential constraints LaValle (1998). This single-query planning technique explores the state space by expanding a tree towards the goal configuration. Different heuristics guide the search towards the goal configuration (Burns and Brock, 2007b; Kuffner and LaValle, 2000). When generating and connecting samples, kinematic and dynamic constraints can be considered (LaValle and Kuffner, 2001). The probabilistic roadmap (PRM) introduced by Kavraki et al. (1996) became a popular and widely used planning framework for static environments, as it is able to efficiently answer multiple path queries even in high-dimensional configuration spaces. Many variants of the original framework have been presented, for instance, regarding sampling strategies (Boor et al., 1999; Guibas et al., 1999), or node connection strategies (Amato et al., 1998; Bohlin and Kavraki, 2000). Since in real-world applications, the assumption of a perfect world model does not hold in general, Burns and Brock (2007a) proposed the predictive roadmap planner, which incorporates uncertainty into the roadmap and the planning process. To overcome the assumption of a static world, the elastic roadmap framework of Yang and Brock (2006) allows for the movement of nodes and for updates of connectivity information in the roadmap to adapt to changes in the environment. In contrast to the assumption of rigid environments, we consider obstacles to be deformable in our approach.

7.2.2 Motion Planning with Deformable Objects

Recently, considering physical properties of robots and environments, for instance in terms of their deformation properties has received increased attention. In this context, different planners for deformable robots have been developed. The works of Kavraki et al. (1998), Anshelevich et al. (2000), and Bayazit et al. (2002) mark first steps in this direction. These approaches share the idea that a probabilistic roadmap is used as underlying motion planner and a deformation simulation is carried out to determine the expected deformations of the robot along a path. The underlying deformation models, however, vary. The robots are modeled as flexible surface patches using a two-dimensional finite element approximation (Holleman et al., 1998), or as volumetric elements using mass-spring systems (Anshelevich et al., 2000). To achieve a computationally more efficient realization, Bayazit et al. (2002) employ geometric free-form deformations in their approach instead of physical models. Gayle et al. (2005) present a motion planning framework for complex environments and apply it to plan the motions of flexible surgical tools that are inserted into statically modeled blood vessels. Their deformation model considers constraints for volume preservation to achieve a physically realistic simulation of deformations similar to the model introduced by Teschner et al. (2004). In contrast to our approach, these planners compute the object deformations necessary to avoid collisions with the environment when generating the roadmap. Planning motions for deformable robots was recently revisited by Mahoney et al. (2010). Similar to our approach, they address the computational demands of accurate deformation simulations during runtime. Their approach is based on the PRM framework and contains both configurations that represent rigid transformations and a set of deformation configurations in a reduced configuration space. The deformed configurations are computed in a preprocessing step and transformed to a reduced configuration space using principal component analysis. The focus in this approach is on speed rather than on physically accurate deformation computations, however, energy constraints can be considered to exclude unrealistic deformation configurations when planning motions. These approaches have in common that the environment is rigid and robots can deform to avoid collisions, whereas our approach allows the robot to deform obstacles in the environment.

An approach to planning in completely deformable environments has been proposed by Rodríguez et al. (2006). They employ the previously discussed mass-spring model with additional physical constraints for volume-preservation of Teschner et al. (2004) to enforce a more realistic behavior of deformable objects and search for a path to a goal location using rapidly exploring random trees. Planning for surgical tools in deformable environments was addressed, among others, by Alterovitz et al. (2009), Maris et al. (2010), and Patil et al. (2011). Maris et al. (2010) plan paths for a surgical tool using a 3D simulation based on a mass-spring model. They optimize the control points of a path with respect to constraints that consider the stiffness of objects and the penetration depth of the tool. Alterovitz et al. (2009), in contrast, plan needle placement in the 2D plane and account for deformations using an finite element simulator similar to ours. Recently, Patil et al. (2011) presented an extension to this work that also incorporates the potential uncertainty during path execution into the planning process and chooses the path with the highest probability of success. Reed et al. (2011) demonstrate the applicability of this approach to steering robotic needles in a real experiment with an ex-vivo liver. The focus in this work, however, is more on the complex kinematics of the steerable needle than on the deformation properties of the environment.

A drawback of the approaches discussed above is that they need to compute the deformation simulations during runtime. This is time-consuming and therefore not desirable when planning the motions of real robots. To overcome this limitation, we introduced a model of the deformation cost function based on Gaussian process regression that reduces the planning runtime by several orders of magnitude. Computationally demanding simulations are carried out in a preprocessing step for each type of object and are independent of the shape of the static part of the environment. However, in contrast to the mentioned approaches, we need to introduce new assumptions: we consider stationary objects only and we ignore potential interactions between different deformable obstacles.

Motion planning in the broadest sense also comprises handling, grasping and manipulating deformable objects. We briefly discuss interesting approaches that address different problems in this context. Grasping of deformable objects is addressed in the work of Gopalakrishnan and Goldberg (2005). They pursue the idea of deform-closure grasps, similar to stable form-closure grasps for rigid objects. When computing two-point deform closure grasps, they consider the mechanical properties of objects using a 2D finite element model to determine the optimal gripping force. Hirai et al. (2001) proposed a control law for grasping and manipulating deformable objects that can deal with uncertainties in the deformation model. The aim of their work is to bring the deformable object into a target configuration by manipulating a set of control points on the object. Planning the manipulation of deformable linear objects was addressed, among others, by Ladd and Kavraki (2004), Saha and Isto (2007), and Moll and Kavraki (2006). Ladd and Kavraki (2004) present an approach for knot untangling by minimizing energy functions. Applications for this problem are, for instance, in studying DNA or protein folding and unfolding procedures. The motion planner of Saha and Isto (2007) computes actions for two cooperative robotic manipulators to tie different types of knots. The planner relies on a topological model rather than a physical model of the deformable objects. Similar to this approach, the planner of Moll and Kavraki (2006) computes paths for deformable linear objects, motivated by the application of surgical suturing. They, however, determine stable configurations of the objects by considering minimal energy curves. Cusumano-Towner et al. (2011) study the problem of bringing clothes into a desired configuration with a two-armed manipulation robot. They employ a strain-limiting finite element model to reason about desirable grasps. In contrast to our approach that considers object deformations as additional costs, these approaches are designed to reason about the deformation of objects to bring them into desired deformed configurations.

7.2.3 Robot Learning with Gaussian Processes

The efficiency of our proposed motion planner derives from the fact that we model the costs introduced by deforming an object with the Gaussian process (GP) framework (Rasmussen and Williams, 2006). In the context of robot learning tasks, GPs are becoming increasingly popular and have been applied to different problems, for instance, to modeling terrain (Lang et al., 2007; O'Callaghan et al., 2010; Vasudevan et al., 2009), learning motion and observation models (Ko and Fox., 2009), or modeling gas distributions (Stachniss et al., 2009). In particular, the approach of Vasudevan et al. (2009) is interesting, as it is able to deal with large-scale data sets. It reduces the number of training samples required for GP regression by organizing the data in a k-d tree and considering only a local neighborhood for the prediction of an unknown query point. This inspired our approach, which requires large training data sets to predict the deformation costs of trajectories. In a similar way, we reduce the number of training samples for the GP to the subset of the most relevant ones for solving the regression problem at hand.

In the context of navigation and path planning, GPs have been used to incorporate uncertain quantities into the cost function that can be only partially observed. Henry et al. (2010), for instance, use GPs to predict human motion behavior to plan more efficient robot trajectories in crowded environments. The approach of Murphy and Newman (2010) uses GPs in a first step to classify different terrain types in aerial images, and subsequently to model a cost function for traversing different terrain types. They obtain a probability density over the terrain costs by combining the probabilities of encountering different terrain types in a given location with their associated traversal costs. In this way, they are able to search for the most likely low cost path in an image.

8 Discussion and Outlook

Autonomous robots that consider deformable objects in their environment can accomplish a larger class of motion planning tasks than robots that treat all obstacles as rigid. In this thesis, we presented several techniques to enable robot motion in the presence of deformable objects. We addressed the acquisition of deformation models, efficient representations for planning, and application of the developed motion planning framework to robots operating in real-world environments with deformable objects. In the following, we summarize the achievements and insights gained in each area and outline directions for future work.

Deformation model learning

We presented a robotic system that is equipped with actuators and sensors to acquire models of deformable objects. Our robot can observe an object from different viewpoints with a depth camera and then generate a geometrical model. To learn about the deformation properties of an object, the robot performs indentation tests. While deforming the object with its manipulator, it observes the surface deformations with the depth camera mounted to its hand, and, in addition, it measures the applied forces with a force sensor integrated into its wrist. So far, the indentation tests are carried out in a fixed experimental setup and we provide the robot with a set of manipulator configurations for observations and interactions. In principle, the presented hardware setup allows for an autonomous exploration and acquisition of deformation models. Such an exploration task could be investigated in a further step. This provides new challenges, for instance, the robot needs to ensure that an object is approached from a reasonable direction, such that it is deformed and not moved away.

The observations obtained during deformation are used to optimize the material parameters of a finite element model with a gradient-based error minimization scheme. The deformation model we use is based on continuum mechanics and describes linearly elastic, homogeneous, and isotropic material. These materials are characterized by two parameters, Young's modulus and Poisson's ratio. Our experiments showed that Young's modulus, characterizing the stiffness of an object, can be estimated with a low variance over different applied forces. The estimation of Poisson's ratio, which describes the volume preservation of an object, exhibits more variance. This could be related to the experimental setup of the camera, which makes it difficult to observe an extension of the object transverse to the applied force. Therefore, the influence of Poisson's ratio on the error function to be minimized is smaller. In several experiments, we demonstrated that the learned models can be used for realistic simulations of object deformations and that deformations as well as forces could be predicted accurately. However, the assumptions underlying the deformation model are not always applicable for the objects under consideration. Effects such as viscoelasticity, nonlinearity, and inhomogeneity could potentially be considered in the future. When thinking of deformable objects that appear in the context of service robotics in domestic environments, items such as plants or cloth come to mind. An elastic deformation model does not apply here. For instance, cloth in fact is two-dimensional and has a high resistance to stretching and shear forces but a low resistance to bending. Deformation models such as the one proposed by Stumpp et al. (2008) could be investigated. Since our parameter optimization scheme is independent of the underlying deformation model, it could be easily extended to account for various deformation models.

Another interesting direction for future research are different forms of interaction with the objects to learn about their deformation properties. In addition to indentation tests, a robot could carry out actions, such as grasping, pulling, or bending the object. Similar ideas were recently pursued in the work of Boonvisut et al. (2012). They considered several robot trajectories to deform elastic materials in different ways. Such an approach could possibly be applied to deform cloth or plant leaves. Furthermore, one could think of deformable objects that can be grasped and manipulated by a robot, such actions could possibly be used to identify material parameters.

Deformation cost functions for planning

The deformation models acquired by the robot can be used in a physical simulation framework to compute interactions between objects and their resulting deformations. We make use of such simulations to evaluate the deformations of obstacles when planning the motions of the robot. To obtain a cost function for planning, we consider the potential energies of objects that are deformed by the robot along a trajectory.

To avoid time-consuming simulations during planning time and to realize an efficient planning system, we introduced deformation cost functions for objects. We assume that objects can be deformed by the robot, but are stationary. Furthermore, we neglect interactions between different obstacles. As a result, the deformation cost function depends on robot trajectories relative to an object and can be determined in a preprocessing step.

We model deformation cost functions of objects using Gaussian process (GP) regression. This nonparametric approach to regression describes a probability distribution over functions based on a set of training data. In our case, the training data is a set of robot trajectories that lead to object deformations and their associated deformation costs. These training examples can be generated in a preprocessing step by carrying out corresponding simulations. Since the GP model has a runtime cubic in the number of training samples and, for high-dimensional trajectory inputs, requires a considerable amount of training data, we presented an efficient realization that uses only local samples to build local GPs. This allows for a substantial speed-up independent of the overall number of samples. We considered two different covariance functions for the GP modeling task and showed in experiments that the nonstationary neural network covariance function is better suited to modeling deformation costs of robot trajectories. Gaussian processes are a state-of-the-art regression technique and are based on a sound probabilistic framework. In addition to providing an estimate for new function values, they model the uncertainty about the prediction. This proves particularly useful in case of incomplete and noisy data. In our case, the function values are provided by simulations, and no noise is expected with respect to the simulation outcome for fixed material parameters and individual trajectories. However, noise can come into play when considering the uncertainty about the estimated material parameters on the one hand, and the function over the space of trajectories on the other hand. For instance, neighboring trajectories could either crumple the object or move it aside, thereby resulting in largely differing function values. Such types of uncertainty about the outcome of a trajectory can be adequately modeled by the GP framework. It would be interesting to incorporate such uncertainties into the cost function in the future.

The motions considered for GP regression so far are restricted to simple linear trajectories. A promising direction for future work could be to investigate to which extent such precomputations can be transferred and can be useful for objects that might be moved during deformation. This might be of particular interest when considering grasp planning. In such a scenario, precomputations of different stable grasps leading to object deformations might be helpful.

Motion planning in the presence of deformable obstacles

We presented a motion planning framework that is based on probabilistic roadmaps and incorporates the deformation costs of motions. The motion planner optimizes the trade-off between motion costs and deformation costs. It determines the deformation costs of path segments in the roadmap using the presented GP-based deformation cost functions. In this way, we are able to efficiently plan motions. Even for manipulation robots with several degrees of freedom, the planning time is in the order of seconds, and therefore by several orders of magnitudes faster than a planner that carries out the deformation simulations online. However, by restricting the deformation cost functions to linear trajectories, we introduce an approximation error. Our proposed motion planner thus trades accuracy for efficiency. In experiments, however, we demonstrated that the planned motions still in general lead to low cost deformations of objects.

In the future, it would be interesting to transfer the motion planning framework to other applications. This could include, for instance, robots with two cooperative manipulators, which are required to manipulate and deform objects, similar to the task of folding laundry, as addressed in the work of Cusumano-Towner et al. (2011). One could also investigate how to incorporate changes to the world that are introduced by the robot, for instance, when it grasps and manipulates objects. Precomputations in such situations still could be helpful, for instance, to identify motions to grasp and manipulate objects in a certain way. When executing the motions, however, the state of the world needs to be tracked, which requires integration of planning and control.

Plan execution

In the context of path execution, we considered a navigation scenario for a wheeled robot. For such systems, collision avoidance based on sensor measurements is essential to avoid unforeseen obstacles such as humans in environments shared between robots and humans. When navigating in environments with deformable objects, the robot needs to be able to distinguish allowed contacts necessary to deform objects along a path from collisions that have to be avoided.

We introduced a probabilistic sensor model that describes the probability of observing a deformable object in a given position and the expected measurements during interaction with a deformable object. During navigation, this sensor model allows the robot to interpret its observations accordingly and to reliably avoid collisions with humans. Since the expected measurements during interaction with a deformable object do not only depend on the robot position, but also on the executed trajectory, the sensor model needs to be learned in advance for trajectories that lead to object deformations. In principle, it is independent of the absolute object position in the environment, similar to object deformation cost functions for planning. It can therefore be learned for each object type in a preprocessing step. Furthermore, it can be restricted to a set of low-cost trajectories that are frequently chosen by the planner and most likely to be executed. In our application, we have determined the corresponding trajectories manually, but in a future step, this selection could possibly be automated.

It would be interesting to investigate an approach to collision avoidance for manipulators. Our manipulator so far executes planned motions in an open loop. This is possible, since its joints can be positioned with high accuracy, and, due to the intimidating appearance of the manipulator, probably no intelligent dynamic obstacle would risk to approach it anyway. The problem of collision avoidance, however, offers interesting challenges for future research. One could investigate, whether a vision or range based approach similar to the one for the wheeled robot could be extended to manipulators. To ensure a safe path execution, possibly integration with an impedance controller could be achieved, if the necessary sensors are available on the robot platform. This could be particularly interesting for more delicate robots that need to be able to apply forces large enough to deform objects and to comply with a planned trajectory, but also small enough to avoid damage to the rigid part of the environment.

8.1 Concluding Remarks

In sum, this thesis presents the key components towards considering physical deformation properties of obstacles when planning robot motions. It integrates learning of appropriate models with motion planning and demonstrates successful application of the developed approaches on different robots operating in real-world environments.

We believe that considering physical properties of the environment is an exciting area of research and has a great potential of improving the scope and applicability of robots. Beyond motion planning, robots can hopefully be enabled to achieve other useful tasks in household and service applications in the future.

Bibliography

R. Alterovitz, K. Goldberg, J. Pouliot, and I. Hsu. Sensorless motion planning for medical needle insertion in deformable tissues. *IEEE Transactions on Information Technology in Biomedicine*, 13(2):217–225, 2009.

N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

E. Anshelevich, S. Owens, F. Lamiraux, and L. E. Kavraki. Deformable volumes in path planning applications. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2000.

K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 9(5):698–700, 1987.

S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies. In *Proc. of the Eurographics*, 1995.

D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proc. of ACM SIGGRAPH*, 1992.

K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, 2nd edition, 1996.

O. B. Bayazit, J.-M. Lien, and N. M. Amato. Probabilistic roadmap motion planning for deformable objects. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2002.

M. Becker and M. Teschner. Robust and efficient estimation of elasticity parameters using the linear finite element method. In *Proc. of Simulation and Visualization*, 2007.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256, 1992.

G. Bianchi, B. Solenthaler, G. Székely, and M. Harders. Simultaneous topology and stiffness identification for mass-spring models based on FEM reference deformations. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2004.

B. Bickel, M. Baecher, M. Otaduy, W. Matusik, H. Pfister, and M. Gross. Capture and modeling of non-linear heterogeneous soft tissue. *Proc. of ACM SIGGRAPH*, 28(3): 1081–1094, 2009.

R. Bohlin and L. Kavraki. Path planning using lazy PRM. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2000.

P. Boonvisut, R. Jackson, and M. C. Çavuşoğlu. Estimation of soft tissue mechanical parameters from robotic manipulation data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proc. of ACM SIGGRAPH*, 2002.

M. Bro-Nielsen. Finite element modeling in surgery simulation. *Proceedings of the IEEE*, 86(3):490–503, 1998.

W. Burgard, A.B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2):3–55, 2000.

S. Burion, F. Conti, A. Petrovskaya, C. Baur, and O. Khatib. Identifying physical properties of deformable objects by using particle filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007a.

B. Burns and O. Brock. Single-query motion planning with utility-guided random trees. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007b.

J. Chadwick, D. Haumann, and R. Parent. Layered construction for deformable animated characters. In *Proc. of ACM SIGGRAPH*, 1989.

T. R. Chandrupatla and A. D. Belegundu. *Introduction to Finite Elements in Engineering*. Prentice Hall, 3rd edition, 2002.

D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The trimmed iterative closest point algorithm. In *Int. Conf. on Pattern Recognition*, 2002.

A. P. C. Choi and Y. P. Zheng. Estimation of Young's modulus and Poisson's ratio of soft tissue from indentation using two different-sized indentors: finite element analysis of the finite deformation effect. *Medical & Biological Engineering & Computing*, 43(2):258–264, 2005.

H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion*. MIT Press, 2005.

T. J. Chung. *Applied Continuum Mechanics*. Cambridge University Press, New York, 1996.

F. Conti, O. Khatib, and C. Baur. Interactive rendering of deformable objects based on a filling sphere modelling approach. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

M. Cusumano-Towner, A. Singh, S. Miller, J. F. O'Brien, and P. Abbeel. Bringing clothing into desired configurations with limited perception. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proc. of ACM SIGGRAPH*, 2001.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959.

I. DiMatteo, C. R. Genovese, and G. R. Kass. Bayesian curve-fitting with free-knot splines. *Biometrika*, 88(4):1055–1071, 2001.

D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

P. Fong. Sensing, acquisition, and interactive playback of data-based models for elastic deformable objects. *Int. Journal of Robotics Research*, 28:630–655, 2009.

D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 1997.

D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

B. Frank, M. Becker, C. Stachniss, M. Teschner, and W. Burgard. Efficient path planning for mobile robots in environments with deformable objects. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008a.

B. Frank, M. Becker, C. Stachniss, M. Teschner, and W. Burgard. Learning cost functions for mobile robot navigation in environments with deformable objects. In *Workshop on Path Planning on Cost Maps at the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008b.

B. Frank, C. Stachniss, R. Schmedding, M. Teschner, and W. Burgard. Real-world robot navigation amongst deformable obstacles. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010a.

B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *Proc. of the Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conference (RSS)*, 2010b.

B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Efficient motion planning for manipulation robots in environments with deformable objects. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011a.

B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Using Gaussian process regression for efficient motion planning in environments with deformable objects. In *Proc. of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*, 2011b.

A. R. Fugl, A. Jordt, H. G. Petersen, M. Willatzen, and R. Koch. Simultaneous estimation of material properties and pose for deformable objects from depth and color images. In *Pattern Recognition - Joint 34th DAGM and 36th OAGM Symposium*, 2012.

R. Gayle, P. Segars, M.C. Lin, and D. Manocha. Path planning for deformable robots in complex environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2005.

S. F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Lab, 1997.

K. G. Gopalakrishnan and K. Goldberg. D-space and deform closure grasps of deformable parts. *Int. Journal of Robotics Research*, 24(11):899–910, 2005.

S. Gottschalk, M. Lin, and D. Manocha, D.and Zeltzer. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM SIGGRAPH*, 1996.

G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

G. Grisetti, R. Kümmerle, C. Stachniss an U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 1999.

E. Guizzo. How Google's self-driving car works. `http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works`, October 2011. Accessed January 3, 2013.

E. W. Hart. Theory of the tensile test. *Acta Metallurgica*, 15(2):351–355, 1967.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics (SSC4)*, 4 (2):100–107, 1968.

M. Hauth and W. Strasser. Corotational simulation of deformable solids. *Journal of WSCG*, 12(1–3):137–145, 2004.

B. Heidelberger, M. Teschner, R. Keiser, M. Müller, and M. Gross. Consistent penetration depth estimation for deformable collision response. In *Proc. of Vision, Modeling, Visualization (VMV)*, 2004.

P. Henry, C. Vollmer, B. Ferris, and D. Fox. Learning to navigate through crowded environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

S. Hirai, T. Tsuboi, and T. Wada. Robust grasping manipulation of deformable objects. In *Proc. of the IEEE International Symposium on Assembly and Task Planning*, 2001.

C. Holleman, L. E. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 4(4):629–642, 1987.

J. Kajberg and G. Lindkvist. Characterization of materials subjected to large strains by inverse modeling based on in-plane displacement fields. *Int. Journal of Solids and Structures*, 41(13):3439–3459, 2004.

M. Kauer. *Inverse Finite Element Characterization of Soft Tissues*. PhD thesis, ETH Zürich, 2001.

M. Kauer, V. Vuskovic, J. Dual, G. Székely, and M. Bajka. Inverse finite element characterization of soft tissues. *Medical Image Analysis*, 6(3):275–287, 2002.

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

L. E. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 1998.

J. Ko and D. Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2000.

R Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013. To appear.

A. M. Ladd and L. E. Kavraki. Motion planning for knot untangling. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 7–23. Springer Tracks in Advanced Robotics, Springer Verlag, STAR 7, 2004.

F. Lamiraux and L. E. Kavraki. Planning paths for elastic objects under manipulation constraints. *Int. Journal of Robotics Research*, 20(3):188–208, 2001.

J. Lang. *Deformable Model Acquisition and Validation.* PhD thesis, University of British Columbia, 2001.

J. Lang, D. K. Pai, and R. J. Woodham. Acquisition of elastic models for interactive simulation. *Int. Journal of Robotics Research*, 21(8):713–733, 2002.

T. Lang, C. Plagemann, and W. Burgard. Adaptive non-stationary kernel regression for terrain modeling. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98–11, Computer Science Department, Iowa State University, 1998.

S. M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. Journal of Robotics Research*, 20:378–400, 2001.

N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press, Cambridge, MA, 2003.

M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2005.

B. Lloyd, G. Székely, and M. Harders. Identification of spring parameters for deformable object simulation. *IEEE Trans. on Visualization and Computer Graphics*, 13(5):1081–1094, 2007.

D. J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, volume 168 of *NATO ASI Series*, pages 133–165. Springer, 1998.

A. Mahoney, J. Bross, and D. Johnson. Deformable robot motion planning in a reduced-dimension configuration space. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In

Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.

B. Maris, D. Botturi, and P. Fiorini. Trajectory planning with task constraints in densely filled environments. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

S. Melax. Dynamic plane shifting BSP traversal. In *Proc. of Graphics Interface*, 2000.

J. Minguez and L. Montano. Nearness diagram navigation (ND): A new real time collision avoidance approach. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2000.

M. Moll and L. E. Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

M. Müller and M. Gross. Interactive virtual materials. In *Graphics Interface*, 2004.

M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. In *Proc. of ACM SIGGRAPH*, 2005.

L. Murphy and P. Newman. Planning most likely paths from overhead imagery. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

R. M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics No. 118. Springer, 1996.

A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. on Visualization and Computer Graphics*, 9(2):191–205, 2003.

S. O'Callaghan, F. T. Ramos, and H. F. Durrant-Whyte. Contextual occupancy maps incorporating sensor and location uncertainty. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

S. Patil, J. van den Berg, and R. Alterovitz. Motion planning under uncertainty in highly deformable environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2011.

F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1993.

K. Pulli. Multiview registration for large data sets. In *Proc. of the Int. Conf. on 3D Digital Imaging and Modeling (3DIM)*, 1999.

E. A. Rakhmanov, E. B. Saff, and Y. M. Zhou. Minimal discrete energy on the sphere. *Mathematical Research Letters*, 1:647–662, 1994.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

K. B. Reed, A. Majewicz, V. Kallem, R. Alterovitz, K. Goldberg, N. J. Cowan, and A. M. Okamura. Robot-assisted needle steering. *IEEE Robotics & Automation Magazine*, 18 (4):35–46, 2011.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Int. Conf. on Neural Networks (ICNN)*, 1993.

S. Rodríguez, J.-M. Lien, and N. M. Amato. Planning motion in completely deformable environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006.

S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of the Int. Conf. on 3D Digital Imaging and Modeling (3DIM)*, 2001.

M. Saha and P. Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

D. S. Schnur and N. Zabaras. An inverse method for determining elastic material properties and a material interface. *Int. Journal for Numerical Methods in Engineering*, 33(10): 2039–2057, 1992.

M. Seeger, C. K. I Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In C. M. Bishop and B. J. Frey, editors, *Proc. of the Ninth International Workshop on Artificial Intelligence and Statistics*. 2003.

R. W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):158, 1984.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, Cambridge, MA, 2006.

J. Spillmann, M. Wagner, and M. Teschner. Robust tetrahedral meshing of triangle soups. In *Proc. Vision, Modeling, Visualization (VMV)*, 2006.

J. Spillmann, M. Becker, and M. Teschner. Non-iterative computation of contact forces for deformable objects. *Journal of Computer Graphics, Visualization, and Computer Vision (WSCG)*, 15(1–3):33–40, 2007.

C. Stachniss, C. Plagemann, and A. J. Lilienthal. Gas distribution modeling using sparse Gaussian process mixtures. *Autonomous Robots*, 26(2-3):187–202, 2009.

A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1994.

T. Stumpp, J. Spillmann, M. Becker, and M. Teschner. A geometric deformation model for stable cloth simulation. In *Proc. of the Fifth Workshop on Virtual Reality Interactions and Physical Simulations (VRIPHYS)*, 2008.

A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, 2005.

D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proc. of ACM SIGGRAPH*, 1987.

M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization (VMV)*, 2003.

M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *Proc. of Computer Graphics International*, 2004.

M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, and W. Strasser. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Journal of Artificial Intelligence*, 128(1–2):99–141, 2001.

G. Van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

S. Vasudevan, F. T. Ramos, E. W. Nettleton, and H. F. Durrant-Whyte. Gaussian process modeling of large scale terrain. *Journal of Field Robotics*, 26(10):812–840, 2009.

L. Verlet. Computer experiments on classical fluids. I. Theromodynamical properties of Lennard-Jones molecules. *Physical Review*, 159(1):98–103, 1967.

C. K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10(5): 1203–1216, 1998.

C. K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 599–621. The MIT Press, 1999.

X. Wu, M. S. Downes, T. Goktekin, and F. Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Computer Graphics Forum*, 2001.

Y. Yang and O. Brock. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation. In *Proc. of Robotics: Science and Systems (RSS)*, 2006.

R. N. Zantout and Y. F. Zheng. Geodesics: a tool for solving material properties inverse problems. In *IEEE Int. Conf. on Industrial Technology*, 1994.