

Learning Object Deformation Models for Robot Motion Planning

Barbara Frank, Cyrill Stachniss, Rüdiger Schmedding, Matthias Teschner, Wolfram Burgard

University of Freiburg, Department of Computer Science, Georges-Koehler-Allee 79, 79110 Freiburg, Germany

Abstract

In this paper, we address the problem of robot navigation in environments with deformable objects. The aim is to include the costs of object deformations when planning the robot’s motions and trade them off against the travel costs. We present our recently developed robotic system that is able to acquire deformation models of real objects. The robot determines the elasticity parameters by physical interaction with the object and by establishing a relation between the applied forces and the resulting surface deformations. The learned deformation models can then be used to perform physically realistic finite element simulations. This allows the planner to evaluate robot trajectories and to predict the costs of object deformations. Since finite element simulations are time-consuming, we furthermore present an approach to approximate object-specific deformation cost functions by means of Gaussian process regression. We present two real-world applications of our motion planner for a wheeled robot and a manipulation robot. As we demonstrate in real-world experiments, our system is able to estimate appropriate deformation parameters of real objects that can be used to predict future deformations. We show that our deformation cost approximation improves the efficiency of the planner by several orders of magnitude.

Keywords: Mobile robots, deformation models, parameter estimation, motion planning, robot navigation

1. Introduction

Perceiving the surroundings and modeling the environment is an important competence of intelligent mobile robots since such models are required for efficiently solving other high-level tasks. For instance, generating a collision-free path through the environment in an efficient way requires path planning, which in turn builds on top of a model of the environment. There exists a variety of approaches for robots to autonomously generate an appropriate model of the environment by addressing the simultaneous localization and mapping problem [1, 2, 3, 4], by autonomous exploration [5, 6, 7], or by addressing both problems jointly [8, 9].

In order to plan motions in learned environment models, the majority of path planning approaches assumes that the environment contains only rigid obstacles [10, 11, 12], although there are a few notable exceptions such as the works of [13, 14, 15, 16, 17]. In reality, not all obstacles are rigid. In domestic environments – a key target domain for service robots – a robot must deal with many deformable objects such as plants, curtains, or cloth. Considering that an object such as a curtain is deformable can enable a robot to accomplish navigation tasks that otherwise cannot be carried out.

To consider deformable objects in the path planning process, such objects need to be handled in a simulation system underlying the planner. The realistic simulation of object deformations is still an active area of research with a variety of relevant applications in computer graphics, virtual reality, games, movies, but also in robotics [18, 19], and medical simulations [20, 21, 22]. In most applications, the underlying parameters for an appropriate deformation simulation are adjusted manually until the results appear visually plausible. This might be applicable for computer games or movies, but does not necessarily lead to a physically realistic computation of the involved forces. These forces, however, need to be known accurately for navigation in the presence of deformable objects. For example, whenever a robot interacts with real-world objects, only limited forces should be applied to them. This is of utmost importance in medical applications or in domestic settings, for instance, whenever robots have to manipulate plants or clothes. Particularly in these domains, robots need exact knowledge about the parameters of the deformation process.

In this paper, we present a complete robotic system that is able to perceive the environment and model the

deformable objects in the scene. The system estimates the deformation properties of objects, and finally is able to plan a trajectory through the environment, taking potential object deformations into account.

Estimating the elasticity parameters of objects not only involves observing and reconstructing the three-dimensional surface of an object. Physical interaction with the object under consideration is required to learn about its behavior when exposed to external forces. Therefore, we equipped our robot with a force sensor at the end of the manipulator and with a depth camera. Based on the observed surface deformations and corresponding forces, our approach seeks to determine the elasticity parameters of the object. This is done by simulating the object deformation under the applied forces using a linear finite element model. An error minimization approach is applied to iteratively adapt the deformation parameters such that the difference between the real object under deformation and the simulation is minimized. As we will demonstrate in the experimental section of this paper, our approach is able to find elasticity parameters that enable our robot to accurately predict the deformations of real-world objects.

Furthermore, we address the problem of planning motions for robots navigating in environments with deformable obstacles and to adequately consider the costs of object deformations. In this context, we present an efficient approximation of the deformation cost function of objects. Throughout this paper, we assume that the robot can deform the objects but does not move them in the environment. This allows us to generate a set of trajectory samples in a pre-processing step and to predict the costs of new trajectories by applying efficient Gaussian process regression. Using this regression approach, the robot is able to efficiently plan trajectories in the presence of deformable objects without the need for time-consuming simulations during runtime. In different experiments, we demonstrate that our approach yields accurate estimates and, at the same time, allows for efficient planning of trajectories along which the robot interacts with deformable objects.

This paper is organized as follows: after discussing related work in the next section, we will give an overview of our planning approach that considers deformable objects and describe the basic principles of the deformation model and the physical simulation underlying our planner in Section 3. In Section 4, we describe how to learn models of deformable objects with our manipulation robot. Next, we present our approach to approximate the deformation cost functions of objects using Gaussian process regression in Section 5. Subsequently, we present two applications of our path

planning system applied to a manipulation robot and a wheeled robot. Finally, in Section 7, we evaluate our system in different experiments.

2. Related Work

2.1. Deformable Modeling and Parameter Estimation

Deformable modeling and parameter estimation are active areas of research. To represent non-rigid objects and to simulate deformations, mass-spring systems have been frequently used. They are easy to implement and can be simulated efficiently [23, 24]. Their major drawback is the tedious modeling as there is no intuitive relation between spring constants and physical material properties in general [25]. Finite element methods (FEMs) reflect physical properties of objects in a more natural way [26]. They are based on elasticity theory and describe object deformations with a small number of physical parameters. Their disadvantage lies in the computational resources required to calculate deformations.

The co-rotational finite element approach [27, 28], which we also use in our current system, avoids non-linear computations and is computationally more efficient. Our system, however, does not depend on the underlying deformation model and therefore, arbitrary approaches can be used in our algorithms. For example, Mousavi et al. [29] employ a principal component analysis as a precomputation step in order to gain efficiency for a minimal loss of accuracy. Similarly, reduced deformable models [30, 31, 32] employ the modal analysis for a more efficient simulation. Other approaches use third order polynomials for this purpose [33].

There are different approaches to determine the physical parameters of models. Bianchi et al. [34] learn the stiffness constants of mass-spring models using a genetic algorithm and comparing it to an FEM reference model. The identification of mass-spring parameters is also discussed in the work of Lloyd et al. [35]. They derive an analytical formulation for the spring parameters from a linear finite element model. Data-driven representations for deformable objects were employed, among others, by Fong [36] and Bickel et al. [37]. Fong [36] 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. [37] represent heterogeneous and nonlinear material. The homogeneous parts of objects, however, are modeled using the linear FEM, similar to our approach.

Different approaches deduce the elasticity parameters of objects 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 [38] determine the material parameters of thin metal sheets including plasticity effects. Choi and Zheng [39] identify Young’s modulus and Poisson’s ratio of soft tissues from indentation tests. Schnur and Zabarar [40] estimate different parameters including Young’s modulus of a two-dimensional nonlinear finite element model. The approach of Becker and Teschner [41], 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.

Estimation of material parameters from 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. [42] 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. Deformation forces are measured with an aspiration instrument operated by a human. Their estimation procedure is designed to operate on two-dimensional image data. Fugl et al. [43] present an approach to determine Young’s modulus and different parameters to model heterogeneous material from observations of deformations due to gravity with an RGB-D camera. Lang et al. [44] collect data of object deformations with a robotic measurement facility, including force sensors and stereo cameras. They model deformable objects as a discrete boundary value problem and estimate Greens’ functions from measured forces and displacements. An interesting approach was recently presented by Boonvisut et al. [45]. In this work, a robotic manipulator performs deformation trajectories, and the parameters are optimized such that a finite element simulation of the trajectory agrees with the observed trajectory. Both systems, however, rely on a complex experimental setup with several external cameras. An alternative approach to tracking deformable objects based on RGB-D data was proposed by Schulman et al. [46]. They present a generative probabilistic model that accounts for occlusions, observation noise, and physical material properties. This model is optimized based on observations of a human manipulating and deforming an object. Since forces cannot be observed, the model is only determined up to a scale factor.

In contrast to most of the previous approaches to parameter estimation, 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.

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 Holleman et al. [47], Anshelevich et al. [13], and Bayazit et al. [14] mark first steps in this direction. They use a probabilistic roadmap (PRM) as underlying motion planner and carry out deformation simulations to determine the expected deformations of the robot along a path. The underlying deformation models, however, vary. Robots are modeled using a two-dimensional finite element approximation [47], or computationally more efficient mass-spring systems [13] and geometric free-form deformations [14]. Gayle et al. [15] present a motion planning framework for flexible surgical tools that are inserted into rigid blood vessels. Their deformation model considers constraints for volume preservation similar to the model introduced by Teschner et al. [23]. Planning motions for deformable robots was recently revisited by Mahoney et al. [48]. Similar to our approach, they address the computational demands of accurate deformation simulations during runtime by precomputing a set of deformation configurations that is considered for planning. In contrast to our approach, these planners consider robot deformations that are necessary to avoid collisions with the environment. An approach to planning in completely deformable environments has been proposed by Rodríguez et al. [18]. They employ a mass-spring model with constraints for volume-preservation [23] 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. [16], Maris et al. [17], and Patil et al. [19]. Maris et al. [17] 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. [16], in contrast, plan needle placement in the 2D plane and account for deformations using a finite element simulator similar to ours. Recently, Patil et al. [19] 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. A reactive approach to robot motion

among deformable clutter such as plants was recently proposed by Jain et al. [49], they introduce a model-predictive controller that, based on whole-body tactile sensing, adapts to deformations and controls the contact force. This approach guides the robot along linear trajectories to a goal, but does not plan complex motions. Furthermore, it requires the robot to be equipped with tactile sensors along its arm.

A drawback of the planning approaches above is that they need to compute deformation simulations during runtime. To allow for an efficient answering of path queries, our approach generates a set of sample object deformations in a preprocessing step and models the costs introduced by deforming an object with the Gaussian process (GP) framework [50]. In the context of robot learning tasks, GPs are becoming increasingly popular and have been applied to different problems, for instance, to modeling terrain [51, 52, 53], learning motion and observation models [54], or modeling gas distributions [55]. Inspired by the approach of Vasudevan et al. [52], we deal with large training data sets by organizing the data in a k -d tree and by considering only a local neighborhood for the prediction of a new query. In the context of navigation, GPs have also been used to incorporate uncertain quantities into the cost function. Henry et al. [56], for instance, use GPs to predict human motion behavior when planning robot trajectories, and Murphy and Newman [57] use GPs to model a cost function for traversing different terrain types.

This paper builds on our previous work [58, 59, 60, 61] and presents a unifying framework for robot motion planning in environments with deformable objects. The system is able to determine appropriate material parameters of obstacles that can be used in physical simulations. We furthermore present an approach to model deformation cost functions that allow for efficient planning for robots operating in 2D and 3D work space.

3. Path Planning considering Deformable Obstacles

In this section, we give an overview of our planning approach and introduce the basic concepts of deformation simulations needed for model learning and planning.

3.1. Overview of our approach

Our approach to motion planning in environments with deformable objects consists of several steps: First, the robot needs to determine an appropriate deformation model of an obstacle. This is done by physical interaction with the object and by measuring the deformation

forces as well as the deformed surface of the object. The elasticity parameters of the object are then determined by optimizing the parameters of a linear finite element model such that it best fits the observations.

When generating a motion plan, such a model 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 alternative trajectories must be evaluated when searching for a path to a specific goal. 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 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.

3.2. Planning using Probabilistic Roadmaps

To plan trajectories for our robots, we use the probabilistic roadmap framework introduced by Kavraki et al. [62]. The key idea is to represent the collision-free configuration space 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. Such a roadmap can be precomputed given a model of the environment. To actually plan a trajectory for the robot, the current robot configuration as well as the target configuration are connected to the graph. Most motion planning systems assign costs to the edges that correspond to their distance in configuration or work space. Then, a standard graph search technique such as A* or Dijkstra’s algorithm can be applied to search for the optimal path between a given starting and goal point in the roadmap.

Since we explicitly allow our robot to interact with deformable objects, we also allow for samples and edges that lead to collisions with such objects when generating the probabilistic roadmap. Accordingly, we need to consider the deformation costs when planning trajectories. Our system uses a weighted sum between the distance of the nodes in configuration space and the defor-

mation costs. For an edge between the nodes i and j , its cost is given by

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

where $\alpha \in [0, 1]$ is a user-defined weighting coefficient. The term $C_{travel}(i, j)$ corresponds to the distance between nodes in configuration space. The term $C_{def}(i, j)$ represents the costs that are introduced when the robot deforms objects along its trajectory and is determined efficiently using GP regression.

3.3. Deformation Simulation

To consider non-rigid obstacles in the environment during planning, we need a model that allows us to compute the deformations given an external force. We first present an overview of our simulation system *DefCol Studio*^{1,2} before we go into details of the underlying deformation model based on the finite element method (FEM) in the next section. Our simulation system uses a tetrahedral mesh to represent deformable objects and proceeds as follows: in each time step, it computes deformations and unconstrained motions of objects, then it detects collisions, computes contact forces for colliding points, and finally computes the resulting deformations from the repulsion forces.

3.3.1. 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. [63]. 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. Since the space is usually filled sparsely and non-uniformly, this method consumes less memory than an explicit discretization. The hash key is computed from the coordinates of the corresponding grid cell. Consequently, only the elements with the same hash key need to be checked for collisions. To check for collisions, intersections between points and tetrahedra are computed. This can be done efficiently using barycentric coordinates of the points with respect to the tetrahedra.

¹B. Heidelberger: DefCol Studio – Interactive deformable modeling framework. <http://www.beosil.com/projects.html#DefColStudio>, last accessed March 18, 2014.

²M. Teschner: Defcol Studio 1.1.0. <http://cg.informatik.uni-freiburg.de/software.htm>, last accessed March 18, 2014.

3.3.2. Computation of Contact Forces

To handle collisions between the robot and deformable objects, we employ the force-based collision handling scheme proposed by Spillmann et al. [64]. It combines the advantages of penalty and constraint-based collision handling schemes. For a set of colliding points of a tetrahedral mesh, a collision-free state is computed using a linearized relation between internal forces and displacements of all affected points. Contact forces can be computed analytically to obtain the collision-free state while conserving overall system energy.

3.4. Deformation Model

A deformable solid can be described by its undeformed state and a set of material parameters that determine how it deforms under applied forces. A deformation is then specified by a displacement field $\mathbf{u} : \mathbf{x}_0 \mapsto \mathbf{x}_0 + \mathbf{x}$, which maps each point \mathbf{x}_0 of the object in its reference position to a deformed position $\mathbf{x}_0 + \mathbf{x}$. Elasticity theory provides the corresponding constitutive equations. Since we restrict ourselves to linearly elastic, homogeneous and isotropic material, we employ a linear relation between stress $\boldsymbol{\sigma}$ and strain $\boldsymbol{\epsilon}$ given by the generalized Hooke's law:

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\epsilon}. \quad (2)$$

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 \mathbf{u} , and external forces acting on the object, we consider the total potential energy Π of a solid, which is given by

$$\Pi = U + WP. \quad (3)$$

Here, WP is the work potential, which is determined by the external forces acting on an object. The inner or elastic energy U is given by

$$U = \frac{1}{2} \int_V \boldsymbol{\sigma}^T \boldsymbol{\epsilon} dV. \quad (4)$$

A stable equilibrium configuration of a deformation can be found by minimizing the potential energy, which is done by setting the derivatives to zero.

3.4.1. Elasticity Parameters

For linearly elastic and isotropic material, the matrix \mathbf{C} in Eq. 2 depends only on two independent elasticity parameters, Young's modulus E and Poisson's ratio ν [65]. 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:

$$E = \frac{\sigma}{\epsilon} = \frac{F/A}{\Delta x/x} = \frac{Fx}{A\Delta x}. \quad (5)$$

Its unit is force per area and it is frequently specified in $\frac{\text{N}}{\text{dm}^2}$.

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:

$$\nu = -\frac{\epsilon_{\text{trans}}}{\epsilon_{\text{axial}}} = -\frac{\epsilon_y}{\epsilon_x} = -\frac{\epsilon_z}{\epsilon_x}. \quad (6)$$

Since we consider isotropic material, the changes in the two directions y, z perpendicular to the direction x of the applied force are equal.

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.

3.4.2. Finite Element Approximation

We use the finite element method (FEM) to approximate the deformation of a continuous object. The key idea is to discretize the object into a finite set of volumetric primitives, tetrahedrons in our case, and to compute the deformations inside the elements by an interpolation using the nodal values. First of all, the displacement field inside a tetrahedron is approximated using the displacements of the nodes. To compute the strain ϵ from the nodal deformations in our model, we use the linear Cauchy strain tensor, which is efficient to compute:

$$\epsilon_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \quad (7)$$

The strain in terms of the displacements \mathbf{q} of the nodes can then be written as a matrix multiplication $\epsilon = B\mathbf{q}$, where B expresses the partial derivatives from Eq. 7

in terms of the nodal displacements using linear shape functions. The inner energy U_e of a tetrahedral element e can then be written as

$$U_e = \frac{1}{2} \int_e \epsilon^T C^T \epsilon dV.$$

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

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

where V_e is the volume of e . When we define the element stiffness matrix to be $K_e := V_e B^T C^T B$, we obtain

$$U_e = \frac{1}{2} \mathbf{q}^T K \mathbf{q}. \quad (8)$$

To find an equilibrium configuration of the deformable object, we minimize the total potential energy by setting the partial derivatives of Π with respect to the displacements q_i to zero.

The derivatives of U_e with respect to q_i result in $\frac{\partial U_e}{\partial q_i} = (K_e \cdot \mathbf{q})_i$ and describe the elastic forces acting on the nodes of the model. As we only consider the point loads f_i in the work potential, the partial derivatives with respect to the displacements q_i are given by $\frac{\partial WP}{\partial q_i} = f_i$. Therefore, setting the derivative of the potential energy $\frac{\partial \Pi}{\partial \mathbf{q}}$ to zero leads to $K_e \cdot \mathbf{q} - \mathbf{f} = 0$. 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 can be written as

$$\mathbf{F} = K\mathbf{Q}. \quad (9)$$

Using linear shape functions and the linear Cauchy strain tensor for the computation of the strain leads to problems, as the linearization assumption is only valid close to the equilibrium. Furthermore, this tensor is not rotationally invariant. This leads to ghost forces, which result in distortions for large rotational deformations. To account for that, we use the co-rotational finite element formulation of Hauth and Strasser [66] and Müller and Gross [28] and keep track of the rigid body motion for each element by extracting the rotation from the transformation matrix using polar decomposition. Applying the strain tensor in the rotated frame leads to rotational invariance and has low computational costs compared to the nonlinear strain tensor.

The deformation model is interesting for us for two reasons: first, we want to estimate the elasticity parameters E and ν of deformable 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.

4. Learning Deformation Models

Modeling the deformation behavior of real objects requires interaction with them to measure the forces as well as the resulting deformations. In this section, we introduce our approach to learn deformation models of real objects with a manipulation robot. The key idea is to compare the observations of the robot to a finite element simulation. The observation of the force allows us to establish the force-displacement relation from Eq. 9. In this way, we are able to estimate the parameters of the stiffness matrix by minimizing the error between observed and simulated deformation.

4.1. Data Acquisition

Our system for acquiring data of deformable objects consists of a mobile platform with a 7-DoF manipulator that is equipped with a force-torque sensor and an RGB-D camera (see Figure 1a). This setup allows the robot to observe objects from different view points, to acquire point clouds of their surfaces, to deform them, and to measure the corresponding deformation forces in a flexible way. In principle, before deforming an object, the robot needs to decide, whether or not it is safe to deform the object without destroying it. This task is not explicitly handled in our approach. We assume that the robot can probe to deform the object without directly destroying it.

The manipulator consists of five Schunk Powercube modules and a 2-DoF hand. These modules have a high repeat accuracy of 0.02 degrees and therefore allow for an accurate estimation of the robot’s position. The deformation forces are measured 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. Furthermore, a Microsoft Kinect RGB-D camera using the structured-light measurement principle is attached to the robot’s hand. This allows the robot to obtain 3D measurements of the object under investigation.

4.1.1. Geometric Models for Simulation

The finite element simulation requires a volumetric model of an object. Such a model can be computed from a surface mesh of the object. Thus, the robot first acquires a 3D surface representation of the object by observing it from different viewpoints and by registering the corresponding point clouds into a consistent model. The task of a registration algorithm is to align overlapping scans of the same object, that is to compute a translation and a rotation that align the surfaces correctly. In our approach, we apply the iterative closest point (ICP) algorithm by Besl and McKay [67], with some extensions similar to the ideas given by Pulli [68] and Rusinkiewicz and Levoy [69]. For known correspondences, the transformation can be computed in closed form [70]. In general, however, the correspondences are not known. Thus, the ICP algorithm determines correspondences, for instance, using a nearest-neighbor data association, computes a transformation that aligns the scans for these correspondences, and iterates this process. Typically, this procedure converges to a minimum and yields an accurate alignment if a reasonable initial configuration is chosen. In our case, the robot poses from which the scans are recorded provide a reasonable initial guess.

From the registered point clouds, we then generate a triangular surface mesh which in turn is used to determine the volumetric tetrahedral mesh. To construct this tetrahedral mesh, we use the approach of Spillmann et al. [71]. This approach first computes a signed distance field, in which voxels having a negative sign represent the volume of the object. In a second step, the spatial domain is divided into a uniform axis-aligned grid. All cells of 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 the 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. The individual steps of this reconstruction procedure are illustrated in Figure 1. This approach is particularly suited for real-world data, as it can handle unorientable, non-manifold, and even incomplete data.

In some situations, it is not possible for the robot to observe the object from all necessary viewpoints in order to obtain a closed surface mesh. This might be the case when the object is partially occluded or parts of the workspace are not accessible to the robot, e.g. in case the object is sitting in front of a wall or on a table. To obtain a closed surface mesh that clearly limits

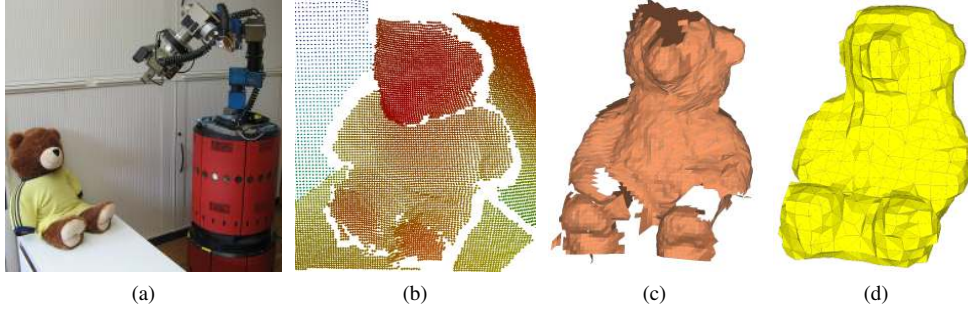


Figure 1: Object reconstruction: (a) the robot observes a deformable teddy bear, (b) a point cloud obtained with the RGB-D camera, (c) the surface mesh constructed from four different point clouds, and (d) the tetrahedral mesh computed from the surface mesh.

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. This allows us to generate a model almost from scratch without much overhead for exploration and moving. Our experiments show that a complete model is not needed to estimate the deformation parameters – a partial model is sufficient. The outcome of this step is a complete geometric and volumetric object representation with a known transformation relative to the robot. In our simulator, we perform all deformation computations based on the tetrahedral mesh. The 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.

4.1.2. Deformation of Objects

In our experimental setup, the object is placed on a table in front of the robot and the robot probes the object by moving its end effector downward, in the direction perpendicular to the table surface (Figure 2). This setup guarantees that the object is fixed between the table and the robot, therefore the measured forces correspond to deformations only, not to translations of the object. Furthermore, 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, 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 probing pro-

cedure 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 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.

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 $\mathbf{f}_t \in \mathbb{R}^3$ 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

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.

The governing equation solved by the FEM approximation (Eq. 9) relates the applied forces \mathbf{F}^{ext} and resulting displacements \mathbf{Q} by a stiffness matrix $K(E, \nu)$ that depends on E and ν (Eq. 9). The inverse problem we intend to solve can be stated as determining the stiffness

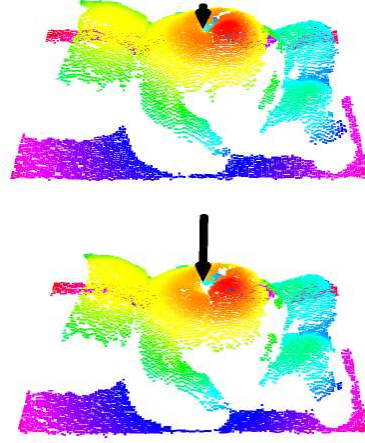
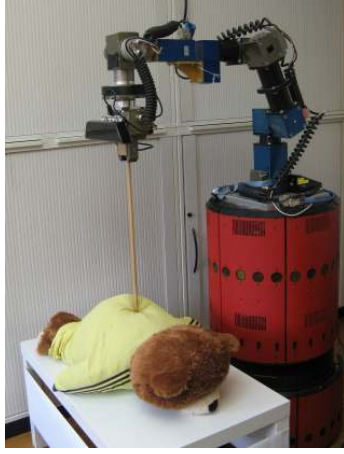


Figure 2: Deformation of an object: experimental setup (left) and two example measurements (right). The surface points are colored according to their depth and the magnitudes of the measured forces are indicated by the arrows.

matrix $K(E, \nu)$ that explains the relation between measured force $\mathbf{F}_{\text{meas}}^{\text{ext}}$ and measured displacement \mathbf{Q}_{meas} :

$$\min_{(E, \nu)} \|K(E, \nu)\mathbf{Q}_{\text{meas}} - \mathbf{F}_{\text{meas}}^{\text{ext}}\|_2^2. \quad (10)$$

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 the 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 observed displacements and minimize their difference:

$$\min_{(E, \nu)} \|\mathbf{Q}_{\text{meas}} - \mathbf{Q}_{\text{sim}(E, \nu)}\|_2^2. \quad (11)$$

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 minimized.

4.2.1. FEM Simulation

We initialize the simulation with the tetrahedral model \mathcal{M} and the corresponding surface points P of an object resulting from Section 4.1.1. Additionally, the stiffness matrices $K(E, \nu)$ of the model elements are computed using given parameters E, ν . 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. 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, that is the mass point on the tetrahedral mesh closest to the contact point. Then, we define $\text{FEMSim}(\mathcal{M}, \mathbf{c}_t, \mathbf{f}_t, E, \nu)$ 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 $P_{E, \nu}$. The deformation for a given force and contact point is governed by the material parameters E and ν of the object.

4.2.2. Error Function

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 an ICP registration procedure to eliminate the effects of small rotations and translations not leading to object deformations as well as inaccuracies in the global position estimation of the model with respect to the robot.

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 of the surfaces:

$$\text{Err}(P_{E, \nu}, P_t) = \frac{1}{|P_t|} \sum_{i \in P_t} \min_{j \in P_{E, \nu}} \|i - j\|^2, \quad (12)$$

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 that considers only a fraction of point correspondences. Otherwise, we would possibly ignore the region of interest in which the object is deformed, due to large point-to-point distances and the error function would not be informative.

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 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 back-propagation (Rprop) update rule that was introduced by Riedmiller and Braun [72] 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. 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. 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.

5. Deformation Cost Functions for Planning

When planning robot motions, we want to consider the costs of object deformations that are introduced by the robot. To achieve this, we first define a measure for such deformation costs that can be obtained by means

of physical simulation of the corresponding robot trajectory. Next, we will introduce the concept of object-dependent deformation cost functions that can be pre-computed under certain assumptions and speed up the planning process. Finally, we present our approach to model the deformation cost function based on Gaussian process regression.

5.1. Deformation Costs of a Robot Trajectory

To measure the cost that 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. 8. The elastic energy of an object corresponds to its deformation and measures its distortion. According to the law of conservation of energy, it directly corresponds to the energy the robot has to expend. In case of an undeformed object, the elastic energy is zero. Otherwise, the elastic energy increases depending on the deformation of the object. Obviously, more energy must be expended for stiffer objects, which is encoded in the object's material parameters and the stiffness matrix K in Eq. 8. For an object O consisting of tetrahedral elements $\{e_i\}$, we define the total inner energy U_O induced by a robot r in a given configuration ζ to be the sum over the inner energies of all elements e_i of the object: $U_O(r, \zeta) := \sum_{e_i \in O} U_{e_i}(r, \zeta)$.

For any given robot configuration ζ , we determine the total deformation costs $C_{def}(\zeta) := \sum_{O \in \mathcal{W}} U_O(r, \zeta)$ by summing over all objects O in our workspace \mathcal{W} . The robot configuration ζ has to be taken into account, since objects might deform differently depending on the configuration of the robot as well as on the history of configurations.

The total deformation costs of a path Γ in our environment naturally 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 :

$$C_{def}(\Gamma) = \sum_{\zeta_i \in \Gamma} C_{def}(\zeta_i). \quad (13)$$

In sum, the deformation costs of a trajectory depend on the sequence of robot configurations during path execution and on the material properties of objects, characterized by Young's modulus E and Poisson's ratio ν .

5.2. Object Deformation Cost Functions

In our definition above, we have seen that the deformation costs are a function of the robot trajectory and

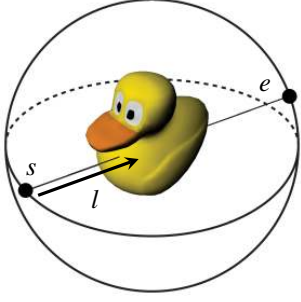


Figure 3: Parametrization: the linear trajectory is described by starting point s and end point e on a virtual sphere around the deformable object. Additionally, we consider the traveled distance l along the trajectory.

the objects on the way. When the planner evaluates trajectory hypotheses, it could in principle carry out deformation simulations for each hypothesis online. This, however, is not desirable in practical applications, since FEM simulations are time-consuming and typically a lot of path hypotheses need to be evaluated. Another possibility is to precompute the deformation costs of edges when generating a roadmap. This saves computation time when answering path queries, but has the disadvantage that recomputations are necessary whenever the environment changes, for instance, when objects are moved.

Our approach is different: we restrict ourselves to an environment, in which objects can be deformed by the robot, but cannot be moved. Furthermore, we ignore interactions between different objects. Thereby, we can 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 frequently.

The idea is to generate some trajectory samples relative to an object and perform the corresponding FEM simulations in a preprocessing step. The problem of estimating the deformation costs introduced by a robot given this set of training samples can then be efficiently approached by regression techniques. Let $y_{1:n}$ be the set of deformation cost values obtained from n simulations, in which the virtual robot executed n different trajectories $\mathbf{x}_{1:n}$. Then, the goal is to learn a predictive model $p(y_* | \mathbf{x}_*, \mathbf{x}_{1:n}, y_{1:n})$ for estimating the deformation costs y_* given a new query trajectory \mathbf{x}_* .

In theory, all possible trajectories through a de-

formable object can be executed. To bound the complexity of the regression problem, we consider only straight line motions through the object here. 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 robot. Furthermore, we take into account the distance l from the starting point that describes the length of the motion. Figure 3 illustrates this parametrization. The points s and e are each described by an azimuth ϕ and an elevation angle θ . 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.

5.3. Modeling Deformation Cost Functions using Gaussian Processes

We approach the problem of estimating the deformation costs of a robot trajectory by means of non-parametric regression using the Gaussian process (GP) model [50]. In this Bayesian approach to non-linear regression, one places a prior on the space of functions using the following definition: A GP is a collection of random variables, any of which have a joint Gaussian distribution. More formally, if we assume that $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $y_i = f(\mathbf{x}_i)$ are samples from a GP and define $\mathbf{y} = (y_1, \dots, y_n)^T$, we have

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}), \quad \boldsymbol{\mu} \in \mathbb{R}^n, \mathbf{K} \in \mathbb{R}^{n \times n}. \quad (14)$$

For simplicity, we set $\boldsymbol{\mu} = \mathbf{0}^3$. The interesting part of the GP model is the covariance matrix \mathbf{K} . It is specified by $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ using a covariance function k . Intuitively, the covariance function specifies how similar two function values $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ are depending on their inputs \mathbf{x}_i and \mathbf{x}_j . A popular choice is the squared exponential covariance function, which is given by

$$k_{SE}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \boldsymbol{\Sigma} (\mathbf{x}_i - \mathbf{x}_j)\right). \quad (15)$$

Here, $\boldsymbol{\Sigma} = \text{diag}(\ell_1, \dots, \ell_d)^{-2}$ is the length-scale matrix and σ_f^2 the signal variance. These parameters together with the global noise level σ_n are known as the hyperparameters of the process.

We furthermore consider in our experiments the neural network covariance function [73, 74, 75], which is

³The expectation is a linear operator and for any deterministic mean function $m(\mathbf{x})$, the Gaussian process over $f'(\mathbf{x}) := f(\mathbf{x}) - m(\mathbf{x})$ has zero mean.

known to better adapt to non-smooth data and to account for variable smoothness. This covariance function is specified as

$$k_{NN}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \arcsin \left(\frac{\beta + 2\mathbf{x}_i^T \Sigma \mathbf{x}_j}{\sqrt{(\beta + 2\mathbf{x}_i^T \Sigma \mathbf{x}_i)(\beta + 2\mathbf{x}_j^T \Sigma \mathbf{x}_j)}} \right), \quad (16)$$

with a bias factor β and Σ, σ_f as defined above.

Learning a GP model is equivalent to determining the hyperparameters of the covariance function that best explain the training data points. This is formulated as an optimization problem by maximizing the marginal log likelihood of the data given the model. We use a standard gradient optimization approach to find the best hyperparameters for a given dataset. More details on the problem formulation can be found in the work of Rasmussen and Williams [50].

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}_* . Let $\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_n]^T$ be the matrix of the inputs. We obtain the predictive distribution $p(y_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ for a new observation \mathbf{x}_* that is again Gaussian with mean

$$\mathbb{E}[f(\mathbf{x}_*)] = k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \right]^{-1} \mathbf{y} \quad (17)$$

and variance

$$\mathbb{V}[f(\mathbf{x}_*)] = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \right]^{-1} k(\mathbf{X}, \mathbf{x}_*), \quad (18)$$

where \mathbf{I} is the identity matrix and $k(\mathbf{X}, \mathbf{X})$ refers to the covariance matrix built by evaluating the covariance function $k(\cdot, \cdot)$ for all pairs of all row vectors $(\mathbf{x}_i, \mathbf{x}_j)$ of \mathbf{X} . In sum, Eq. 17 provides the predictive mean for the deformation costs when carrying out a motion along \mathbf{x}_* and Eq. 18 provides the corresponding predictive variance.

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; 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. For data sets consisting of thousands of training samples, the approach thus becomes inefficient.

Inspired by the approach of Vasudevan et al. [52], we decompose our regression problem into a number of local ones. For a query trajectory \mathbf{x}_* , we determine its M nearest neighbors from the training data as

$$\mathbf{X}'(\mathbf{x}_*) = [\mathbf{x}'_1; \dots; \mathbf{x}'_M] = \arg \min_{[\mathbf{x}'_1; \dots; \mathbf{x}'_M]} \sum_{k=1}^M d(\mathbf{x}'_k, \mathbf{x}_*), \quad (19)$$

where the distance function $d(\cdot, \cdot)$ computes the great circle distance between starting and end points of the respective trajectory samples.

The M nearest neighbors \mathbf{X}' to the query trajectory \mathbf{x}_* are the training data points that have 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 not part of \mathbf{X}' . In our current implementation, we are able to get appropriate predictions by setting $M = 50$. We experienced that the loss is negligible with respect to larger values of M , at least in all our experiments. Determining the M nearest neighbors to \mathbf{x}_* can be computed efficiently using a k -d tree that is once built from the training data. Thus, queries can be obtained in logarithmic time in the number of training examples and the GP prediction does not depend on the size of the training set anymore but only on M .

6. Applications on Real Robots

In this section, we present two applications of our proposed motion planning system. First, we discuss how to plan motions in 3D workspace for a manipulation robot. Second, we present an implementation on a wheeled robot that operates in the 2D plane. In this case, we additionally address the problem of collision avoidance.

6.1. Arm Planning in 3D

In our first application scenario, we plan motions for our robotic manipulator presented in Section 4, which has 7 degrees of freedom in its arm. When constructing the roadmap, we uniformly sample nodes from the configuration space of the robot. For each node, we consider the N nearest neighbors and add an edge if there is a straight line path between both nodes that does not lead to collisions with rigid obstacles. In our implementation, N was set to 50. Our current implementation furthermore applies A* to find the optimal

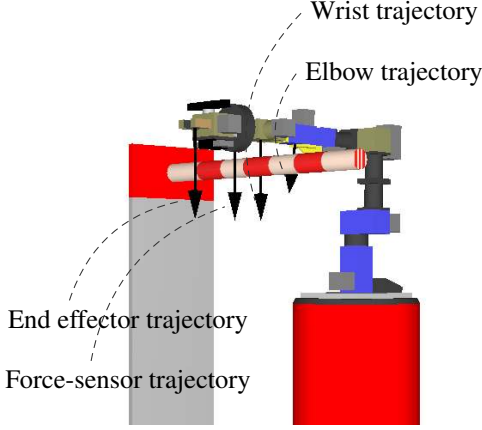


Figure 4: Determining the deformation costs of a manipulator motion: the manipulator moves downward in the presence of the deformable bar. We consider the (approximately linear) trajectories described by its individual body parts and compute their deformation costs using GP regression.

path in the roadmap using the cost function given in Eq. 1. To obtain an admissible heuristic for A^* , i.e., a heuristic that underestimates the real costs specified in Eq. 1, we use the distance to the goal configuration weighted with $(1 - \alpha)$. Thus, we are able to find the path in the roadmap that optimizes the trade-off between travel costs and deformation costs. The deformation costs of edges are approximated using the GP-based regression method introduced in Section 5.

The deformation simulation system considers the motion of a rigid box surrounding the robot’s end effector along the described trajectory to compute the deformation costs. It does not consider the full configuration of the arm. This is clearly an approximation, but it allows us to parametrize the regression problem with a low-dimensional input. Otherwise, the full configuration of the robot would have to be considered in the GP framework. With higher-dimensional inputs, a larger number of training examples is required. Furthermore, the deformation cost function of an object would become dependent on the position of the object relative to the robot.

To account for the fact that not only the end effector, but also other body parts might deform an object, we sample multiple points along the kinematic chain of the robot. Then, we perform the estimation of the deformation costs for the trajectories of all sampled points along the kinematic chain and consider the maximum of the

individual costs

$$C_{def} = \max_b GP(\mathbf{x}_*(b), \mathbf{X}'(\mathbf{x}_*(b)), \mathbf{y}'(\mathbf{x}_*(b))), \quad (20)$$

where b refers to the individual body parts and $\mathbf{x}_*(b)$ to the motion that the body parts carry out given the kinematic structure of the robot, $\mathbf{X}'(\mathbf{x}_*(b))$ and $\mathbf{y}'(\mathbf{x}_*(b))$ are the nearest neighboring trajectories and corresponding deformation costs that are used in the GP regression. Figure 4 illustrates the idea of this approximation, where the trajectory described by the end effector would miss the object, the trajectories of the wrist and the force-sensor, in contrast, would lead to high deformation costs. Considering the maximum in Eq. 20 instead of, for example, the sum, generates more accurate predictions since the largest deformation forces are typically 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.

6.2. Robot Navigation in 2D

In addition to the manipulation robot scenario, we consider autonomous navigation of wheeled robots in the presence of deformable objects. We implemented a navigation system for a wheeled robot, an iRobot B21r platform operating in real environments with deformable objects (Figure 5a). Our application scenario is an office environment with a set of deformable curtains in the corridor. In the context of robot navigation in real environments, not only path planning is an issue; during execution of the planned path, localization and collision avoidance are essential for safe and reliable navigation.

In general, the sensor measurements of the robot are used for collision avoidance, to ensure that the robot never gets too close to an object. When navigating among deformable objects, however, the robot might be required to deform an object, which necessarily leads to collisions with it. Hence, a new challenge arises, that is how to interpret the sensor data of the robot and distinguish measurements corresponding to deformable objects from measurements belonging to rigid and dynamic obstacles that are to be avoided.

A prerequisite to address these issues is an appropriate model of the environment. We use occupancy grid maps to represent static obstacles (Figure 5c) and augment them with information on the deformable objects in the environment (Figure 5b). These models are represented as described in Section 4 and allow us to estimate the deformation costs for moving between grid

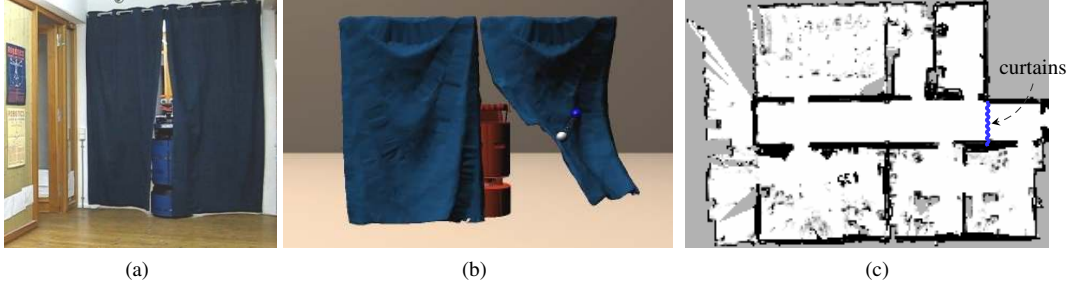


Figure 5: Robot navigation in 2D: our robot Albert in a corridor with curtains (a), the corresponding deformation model (b) and the grid map representing the static part of the world (c).

cells using our GP-based regression approach presented in Section 5.3. The path search is carried out using A* on the grid.

Our implementation is based on CARMEN, a robot sensing and navigation software [76] allowing independent modules to communicate via a middle-ware. It provides modules for low-level robot control and sensing, furthermore modules for path planning, localization, and collision avoidance based on occupancy grid maps. To integrate our approach into CARMEN, we replaced the navigation module with our planner that takes into account deformation costs. In addition to that, we replaced the collision detection method and extended the localization module, which is based on Monte Carlo localization, in a way 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 change of shape during deformation. In the next section, we will address the problem of interpreting the robot’s sensor data for localization and collision avoidance.

6.3. Sensor-based Collision Avoidance for Non-deformable Objects

When navigating autonomously, the robot constantly has to observe its environment in order to react to unforeseen obstacles. At the same time, it might get close to deformable objects when deforming them. Therefore, the main problem in our application is to figure out which measurements correspond to a deformable object, in which case they can be ignored by the collision avoidance system. This section presents our approach to address this problem. By combining the knowledge about objects in the environment and their geometry with estimates of range scans during deformations, we

can reason about the deformability of an observed object.

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, $p(c_i | x, z_i)$ describes the probability that beam i corresponds to a deformable object given the robot position x and the range measurement z_i . Applying Bayes’ formula, we obtain

$$p(c_i | x, z_i) = \frac{p(z_i | x, c_i)p(c_i | x)}{\sum_{c_i, \neg c_i} p(z_i | x, c_i)p(c_i | x)}. \quad (21)$$

Here, $p(z_i | x, c_i)$ is the sensor model for the observation of a deformable object and $p(c_i | x)$ is 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 $p(z_i | x, \neg c_i)$ corresponds to the common sensor model $p(z_i | x)$ when no deformable objects are present.

6.3.1. Learning Sensor Models for Deformable Objects

The sensor model $p(z_i | x, c_i)$ does not only depend on the robot position but also on the trajectory relative to an object. For instance, the robot will measure a different distance to the curtain when it is situated in front of it than it would while passing through and deforming the curtain. Therefore, we determine sensor models corresponding to different trajectories of the robot relative to an object.

For each trajectory, we record different datasets consisting of the robot positions x (provided by the localization module) and the ranges z_i and then manually label the beams reflected by a deformable object. From the labeled measurements obtained along these trajectories,

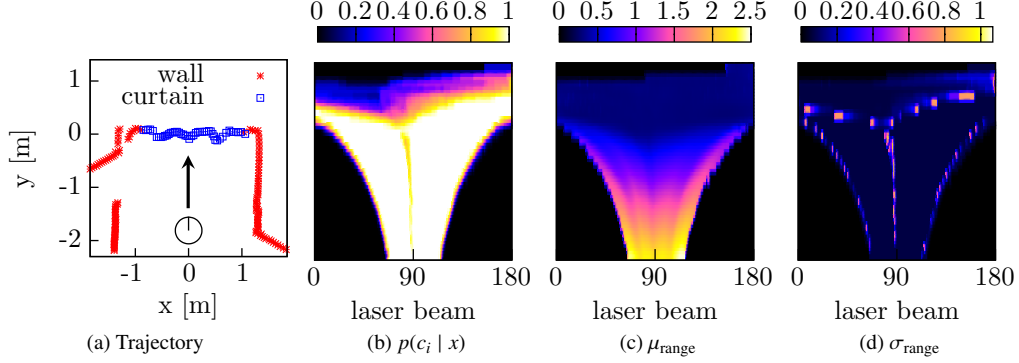


Figure 6: Sensor model for observing the curtain along the trajectory illustrated in (a): shown are (b) the probabilities $p(c_i | x)$, (c) the average beam length, and (d) standard deviation when observing the deformable object.

we compute the statistics

$$p(c_i | x) = \frac{\text{hits}_{def}}{\text{hits}_{def} + \text{misses}_{def}}, \quad (22)$$

where hits_{def} is the number of beams that are reflected by a deformable object and misses_{def} states, how often no deformable object was observed for a given position x and viewing angle i . The sensor model $p(z_i | x, c_i)$ is described by a Gaussian with average range μ and variance σ^2 . An example of such a sensor model for a robot trajectory through the curtain is shown in Figure 6.

6.3.2. Avoiding Collisions

During path execution, the robot constantly monitors its position and also its sensor measurements for utilization in the collision avoidance system. In our case, the robot has to distinguish between allowed collisions with deformable objects and impending collisions with unforeseen or dynamic obstacles, which have to be avoided. This is done by filtering out the range measurements that observe a deformable object with high probability. Therefore, we evaluate Eq. 21 for each beam and identify those beams that can be neglected for the collision avoidance.

This labeling or filtering of the range measurements offers a great potential, since it is done orthogonally to traditional collision avoidance methods. As a result, this technique can be combined with any other collision avoidance technique such as the dynamic window approach [77] or the nearness diagram technique [78].

The detected measurements, which are identified to belong to dynamic obstacles, can be incorporated into the navigation system to update the path of the robot or into any sensor based collision avoidance routine. Our current implementation performs replanning if a path is

blocked by a dynamic object or simply stops the robot if the distance to an obstacle is too close. An example of the collision detection is given in Figure 7.

7. Experiments

In this section, we present evaluations of our approaches to deformation model learning, deformation cost prediction, path planning, and collision avoidance.

7.1. Deformation Model Learning

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.

7.1.1. 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 $E = 100 \frac{\text{N}}{\text{dm}^2}$ and $\nu = 0.3$. The model consists of 625 tetrahedrons and the surface mesh consists of 2,646 points. We 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:

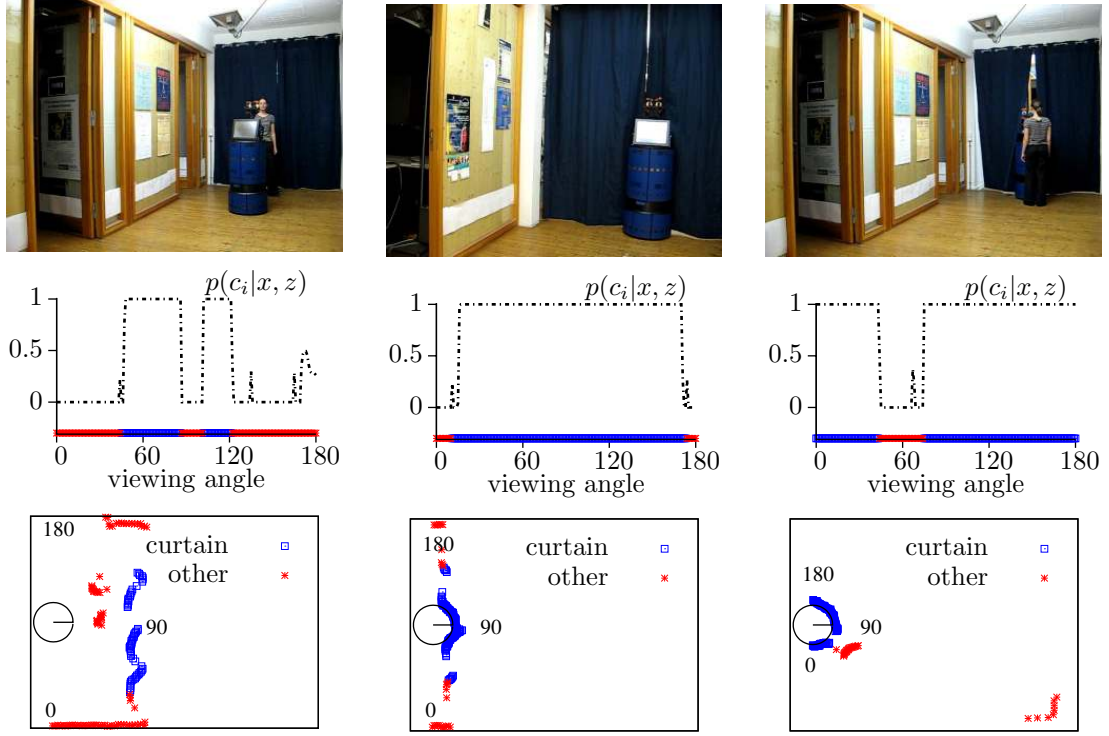


Figure 7: Different collision avoidance scenarios (top row): Laser beams are evaluated with respect to their likelihood of observing a deformable object (second row). The bottom row illustrates the classification of the laser beams.

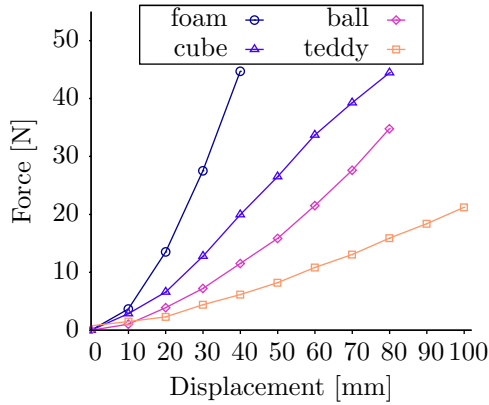


Figure 9: Force-displacement curves for recorded measurements of real objects.

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

$$1 \text{ N}: \sigma_f \sim 0.05|\mathbf{f}| + 1 \text{ N}.$$

- (3) Noise in the estimation of the contact point: $\sigma_c \sim 20$ 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 8 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 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 small, the estimated parameters simply

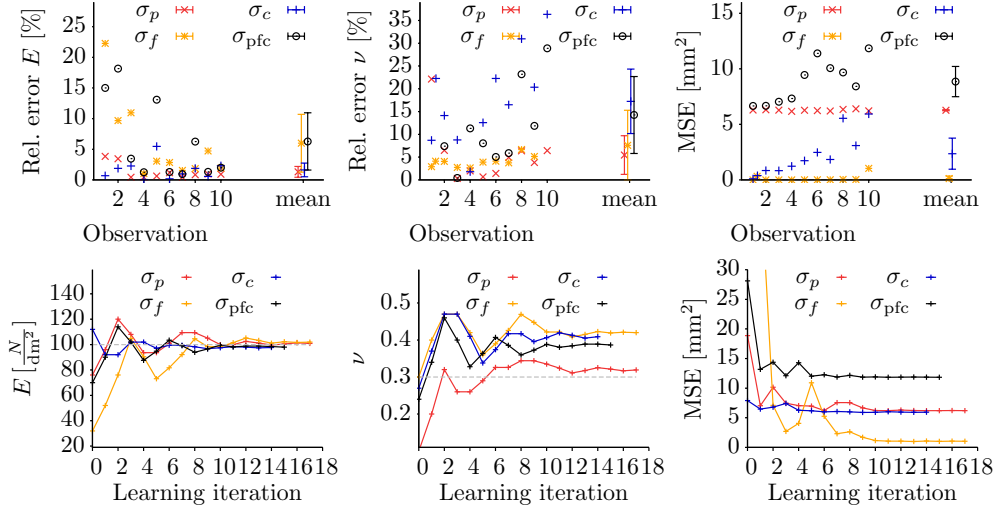


Figure 8: Parameter estimation results for a simulated data set: The plots in the top row show the relative errors in the estimated Young’s modulus E (left) and Poisson’s ratio ν (middle), furthermore the residual MSE of the surface meshes after convergence of the estimation procedure (right). The plots in the bottom row illustrate the evolution of the corresponding quantities in learning runs for one force-deformation sample (with $\mathbf{f} \approx 27$ N).

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.

7.1.2. 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 inflatable 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 different contact points. The force-displacement curves for the recorded samples, with the displacements derived from the manipulator motion are shown in Figure 9. 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 param-

eters and the error in the individual learning runs are shown in Figure 10. 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 on the one side with the nonlinearity in the force-displacement curve, and on the other side with a 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 $340.2 \frac{N}{dm^2} \pm 88.2 \frac{N}{dm^2}$ for Young’s modulus, if we consider 95 % confidence intervals.

The estimation for Poisson’s ratio converges to zero for each sample. This is 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 11. 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 cannot be observed with our sensor setup.

Foam cube: In addition to the foam mat, we ex-

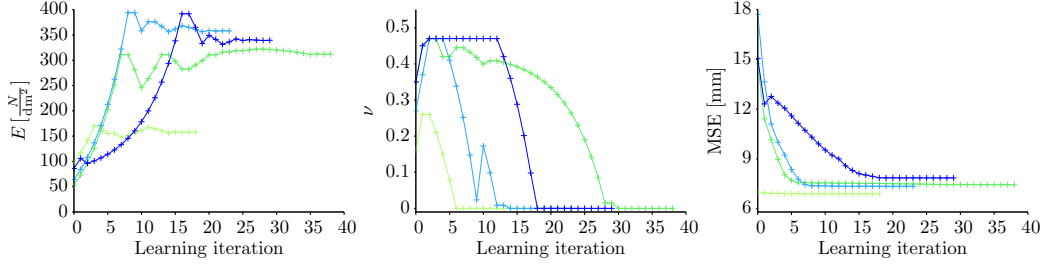


Figure 10: Parameter estimation for the foam mat: The plots show the adaptation of the parameters over the learning iterations (top row) and the MSE of the registered surface meshes (bottom). The colors encode the applied force (green: low force, blue: high force).

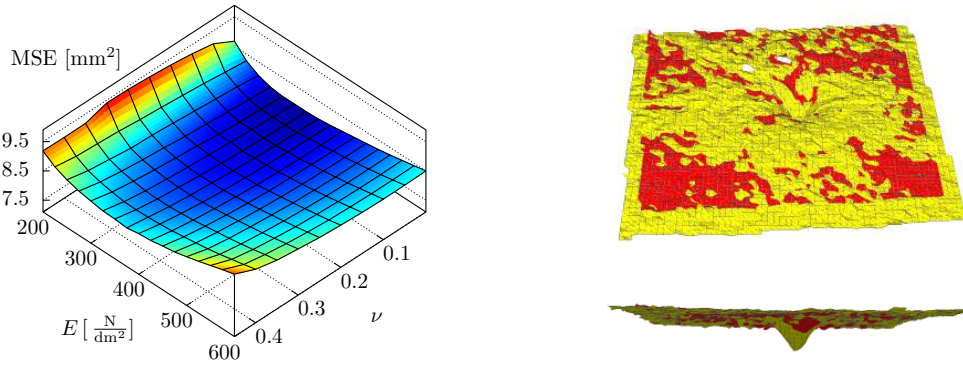


Figure 11: The error function for foam sample 4 ($f = 44$ N) for a uniform sampling of parameters (E, ν): the error varies less with ν than with E . The global minimum at $E = 300 \frac{\text{N}}{\text{dm}^2}, \nu = 0$ agrees with our gradient-based estimation ($E = 339.5 \frac{\text{N}}{\text{dm}^2}, \nu = 0$). The corresponding registered surface meshes are shown on the right.

amined a toy cube consisting of a different type of foam. It is softer, as can be observed from the force-displacement curve (Figure 9). The Young’s modulus estimated for different applied forces varies and is in the range of $148.9 \frac{\text{N}}{\text{dm}^2} \pm 17.2 \frac{\text{N}}{\text{dm}^2}$. Similar to the foam mat, Poisson’s ratio converges to zero, as an extension of the object perpendicular to the camera is hardly observable.

Inflatable ball: The inflatable ball has a large diameter and a small 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. The estimated Young’s modulus is in the range of $65.5 \frac{\text{N}}{\text{dm}^2} \pm 8.1 \frac{\text{N}}{\text{dm}^2}$ 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. 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.

Plush teddy: The plush teddy bear is a 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. We estimated the material parameters for each force-deformation observation and each contact point individually. The results are summarized in Figure 12. For all contact points, the variance in the estimated parameters is lower, if larger forces are applied. This is 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

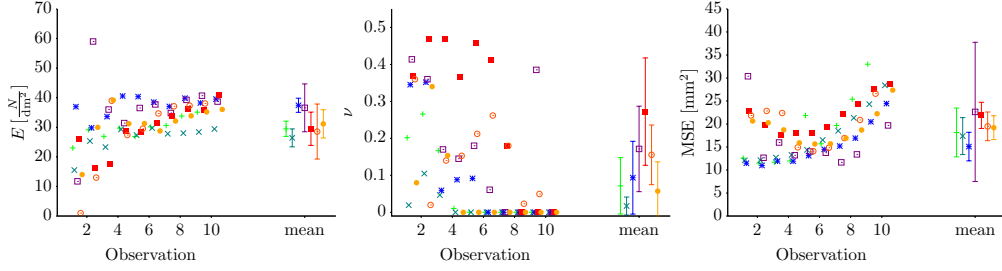


Figure 12: Parameter estimation results for the plush teddy bear for different force-deformation observations on different contact points. The estimated parameters, Young’s modulus (top left) and Poisson’s ratio (top right) and the residual MSE (bottom) are shown for the individual observations, together with the mean and confidence interval for each contact point.

	$\bar{E} \left(\frac{\text{N}}{\text{dm}^2} \right)$	$\bar{\nu}$	MSE (mm)	Force error (%)	$\bar{\nu}$ runtime
Foam (3 samples)	340.2 ± 88.2	0.0 ± 0.0	7.5 ± 0.8	10.0 ± 15.2	2 m 20 s
Cube (7 samples)	148.9 ± 17.2	0.0 ± 0.0	18.2 ± 1.7	12.9 ± 14.0	3 m 7 s
Ball (8 samples)	65.5 ± 8.1	0.27 ± 0.12	15.8 ± 2.7	12.5 ± 17.8	9 m 44 s
Teddy (9 samples)	29.5 ± 3.0	0.07 ± 0.08	18.1 ± 6.0	12.7 ± 10.5	23 m 40 s

Table 1: Parameter estimation results for different real objects. We determined the average over different runs with different forces applied to one contact point.

homogeneous material is obviously not applicable for this object. The residual error for the registered surface meshes tends to increase for larger applied forces, which could be related to the mesh resolution of the underlying volumetric meshes. The parameters estimated in the different experiments, however, are still similar.

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 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 1 lists the aver-

aged 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.

7.2. Deformation Cost Approximation

In this section, we present evaluations of our approach to model deformation cost functions of objects with Gaussian processes. Using the simulation framework described in Section 3.3, 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 motions of a manipulation robot end effector. The trajectory samples were generated by randomly sampling starting and end point on the bounding circle and bounding sphere around the object. An overview of the generated data sets is given in Table 2. In addition to performing evaluations on the generated data sets, we use them in the planning applications in the next section.

In the experiments, we evaluate the accuracy of the predictions using the mean absolute error (MAE) and

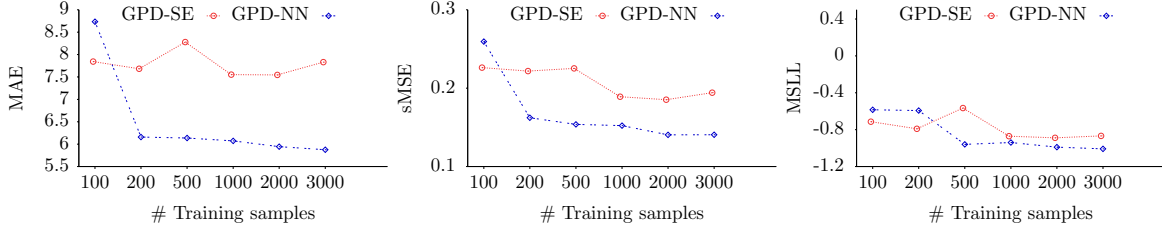


Figure 13: Prediction accuracy of the local GPD models depending on the number of samples used to train the GP hyperparameters. Data set: 3D-Teddy, 5,000 training trajectories were available and the 50 nearest neighbors were used to build the local GPs.

the standardized mean squared error (sMSE)

$$\text{sMSE} = \frac{1}{n} \sum_{i=1}^n \frac{(y_i - \hat{y}_i)^2}{\sigma_{\text{test}}^2}. \quad (23)$$

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 (NLPD) of the true targets:

$$-\log p(y_* | \mathbf{x}_*, \mathcal{D}) = \frac{1}{2} \log(2\pi\sigma_*^2) + \frac{(y_* - \hat{y}_*)^2}{2\sigma_*^2}. \quad (24)$$

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 consider the mean standardized log loss (MSLL)

$$\text{MSLL} = \frac{-1}{n} \sum_{i=1}^n \log p(y_i | \mathbf{x}_i, \mathcal{D}) - \log p_{\text{trivial}}(y_i | \mathbf{x}_i, \mathcal{D}), \quad (25)$$

which is approximately zero for simple models and negative for better models.

We investigate the effects of different optimization strategies and parameters. In particular, 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. The hyperparameters, used in all local GPs, are optimized once on a subset of all samples.

GPO GP decomposition and local optimization strategy, the hyperparameters are optimized for each local GP and test point individually on $M \ll N$ trajectories.

GPF a full GP model using all available training trajectories (if computationally tractable) for hyperparameter optimization and prediction of a test point.

IDW 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.

For the GP models, we furthermore compare two different covariance functions, the squared exponential (GP-SE) and the nonstationary neural network (GP-NN) covariance function.

7.2.1. 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 5,000 trajectories as training data and 300 trajectories as test data. For a fixed number of 50 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 13 and illustrate that training sets

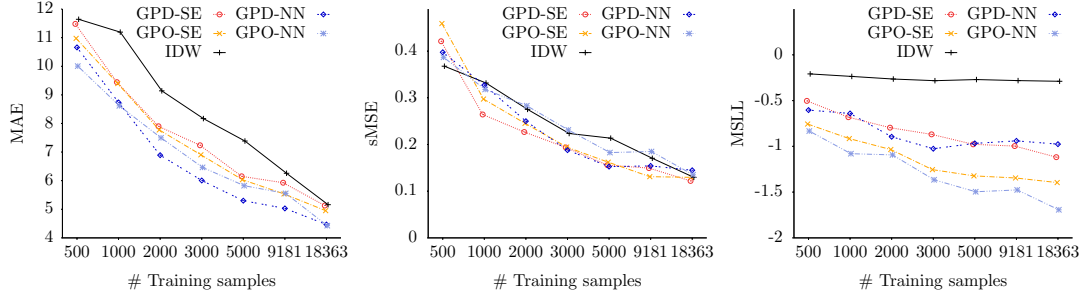


Figure 14: Comparison of different strategies to predict the deformation costs of a robot trajectory: shown are the MAE, sMSE, MSLL depending on the number of training samples. Data set: Foam-3D, 10 nearest neighbors.

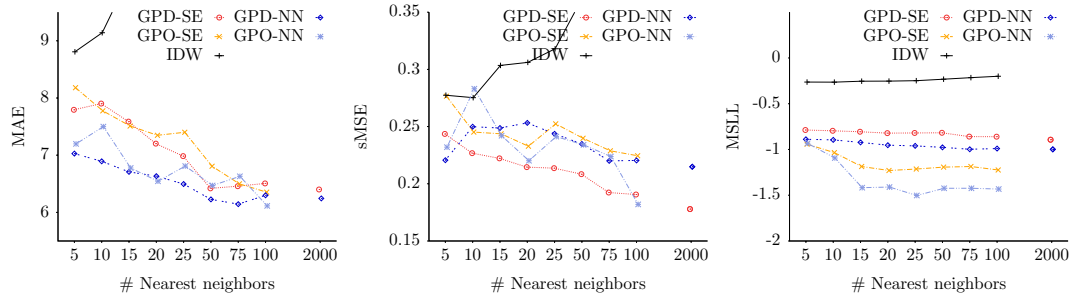


Figure 15: Comparison of different strategies to predict the deformation costs of a robot trajectory: shown are the MAE, sMSE, MSLL depending on the number of nearest neighbors. The full GP models correspond to 2,000 nearest neighbors. Data set: Foam-3D, 2,000 training samples.






Data set		# trajec- tories	# tetra- hedra	run- time
Artificial objects:				
	Duck (2D)	4,284	530	7 h
	Curtain-A (2D)	4,693	500	10 h
Real objects:				
	Curtain-R (2D)	2,035	285	5.5 h
	Foam (3D)	22,950	385	24 h
	Teddy (3D)	12,620	940	24 h

Table 2: Trajectory data sets for different deformable objects: 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.

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 14 for the 3D foam data set. Increasing the number of training samples obviously leads to a smaller

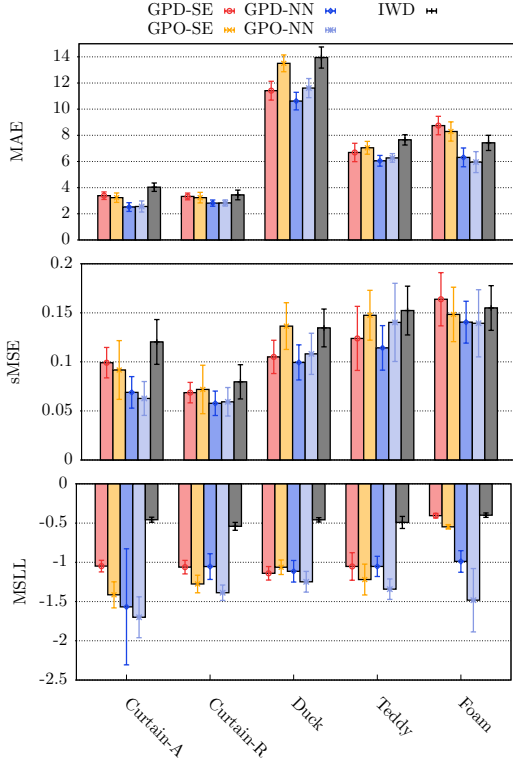


Figure 16: Comparison of the different strategies to predict the deformation costs of trajectories in a ten-fold cross-evaluation.

prediction error for all considered strategies. 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 samples by means of simulations is time-consuming, the training data sets cannot 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 in this experiment. The MSLL, however, is smaller for the locally optimized GPs, which indicates more accurate uncertainty estimates for these strategies.

7.2.2. Number of Nearest Neighbors

In a further experiment, we investigated the influence of the number of nearest neighbors on the pre-

diction accuracy. For a fixed number of 2,000 training samples, 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 15 for the 3D foam data set. Increasing the number of nearest neighbors leads to more accurate predictions and uncertainty estimates for all GP models. With 50 nearest neighbors, the performance is comparable to a full GP model that considers all data points. Using only the M nearest neighbors when evaluating the GP model, however, speeds up computation time, since no computations with large 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 for each GP locally, 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 as the full GP model while significantly reducing computation time.

7.2.3. 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, 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 results 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 16 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. Considering these results, the GPD strategy using the neural network covariance function and optimizing the hyperparameters once on a subset of the available training samples allows for the best trade-off between prediction accuracy and runtime. Thus, we use this strategy in all our planning experiments when determining the deformation costs of the roadmap.

7.3. Motion Planning

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.

7.3.1. Arm Planning in 3D

We evaluated our planning system in two different example applications for our manipulation robot Zora. First, we show a real-world example with the foam mat as a deformable obstacle. This example is designed to close the loop between parameter estimation and motion planning. The robot initially determined the deformation parameters of the object, which in turn allows us to consider the object in simulation and to perform motion planning. Second, we consider a simulated environment with deformable rods. This example is designed to illustrate the advantage of considering deformation costs during planning compared to a planner that ignores deformable objects.

Real-world example: We set up an experimental environment with a deformable foam mat for our manipulation robot Zora (shown in Figure 19). We generated a roadmap that accounts for the static part of the world. The roadmap contains 1,000 configuration samples and 8,635 connections between nodes. 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. The local GPs were built using the neural network covariance function, and the hyperparameters were optimized for a subset of 1,000 trajectories. The deformation costs of edges can be evaluated for the roadmap before answering any queries using our approximation described in

Section 6.1. 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.

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. To illustrate the advantage of considering object deformations when planning motions, we compare our planner to two alternative planners, one that treats them as rigid obstacles, and one that ignores deformable obstacles. 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 (see Figure 17a). The plan generated when completely ignoring the foam mat is shown in Figure 17b. Executing this plan results in tearing down the foam mat, as can be seen in snapshots of the robot motion in Figure 18. The path generated by our planner is visualized in Figure 17c, 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 motion of the manipulator along this path is shown in Figure 19 and in a video that can be found online.⁴

Simulation example: We set up a virtual environment with four deformable rods as obstacles in the workspace of the robot. The rods are modeled to be elastic and fixed to a table and thus can be seen as resembling a construction site scenario with cables or tubes that can be bent. To determine the deformation cost function for the rods, we generated 25,390 training trajectories deforming the rod that allow us to determine the deformation costs of edges in the roadmap using GP regression. Since the four rods have the same deformation properties in this scenario, training samples need to be computed only once. The roadmap in this example contains 2,000 robot configurations and 35,408 edges. We evaluated the deformation costs of all roadmap edges in a preprocessing step. After these pre-computations, we performed seven different planning experiments. We evaluated the planned motions with respect to path length and resulting deformation costs and compared them to a planner that ignores the deformable objects. Results are summarized in Table 3. If deformable objects are ignored, the deformation costs of the executed trajectories are 1.4 to 1718 times larger compared to our planner while the

⁴Real-world experiment: http://www.informatik.uni-freiburg.de/~bfrank/videos/zora_foam.avi

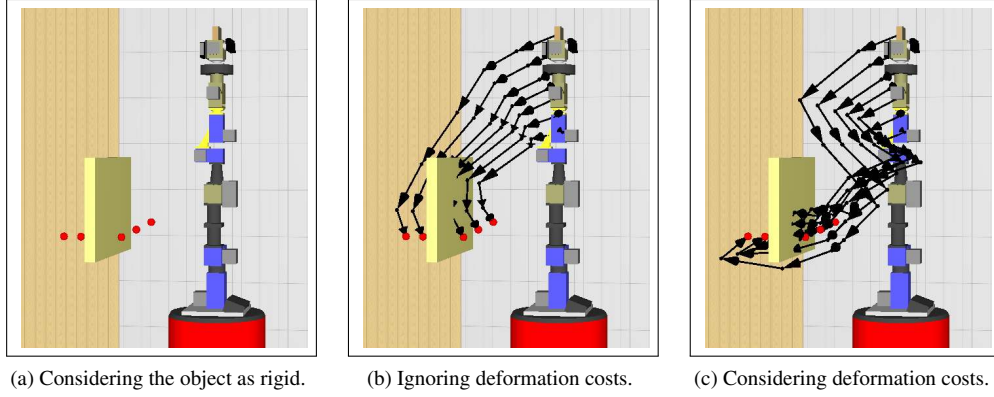


Figure 17: Different motion plans to reach the goal configuration behind the foam mat. (a) A planner treating all obstacles as rigid is not able to find a path to the goal. (b) A planner ignoring deformable obstacles chooses the shortest path. (c) Our planner minimizes the trade-off between motion and deformation costs.



Figure 18: When executing the shortest path that ignores deformable objects (Figure 17b), our robot destroys the experimental setup.



Figure 19: When considering the deformation costs (Figure 17c), our robot keeps the deformation of the foam to a minimum.

trajectories considering deformation costs are 1.1 to 1.7 times longer. A comparison of the planned trajectories is illustrated in Figure 20.

7.3.2. Robot Navigation in 2D

We evaluated our navigation system described in Section 6.2 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 (see Figure 5). 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 described above, the robot is given the task of reaching a goal point beyond the curtains. The planner optimizes the weighted sum of travel costs and deformation costs when search-

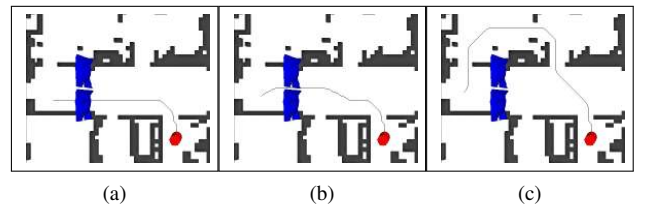


Figure 21: Planning a path for different weightings of the deformation costs: for $\alpha = 0$, the deformation costs are ignored (a), for $\alpha = 0.2$, a longer trajectory is chosen to minimize deformations (b), for $\alpha = 0.8$, deformations are avoided (c).

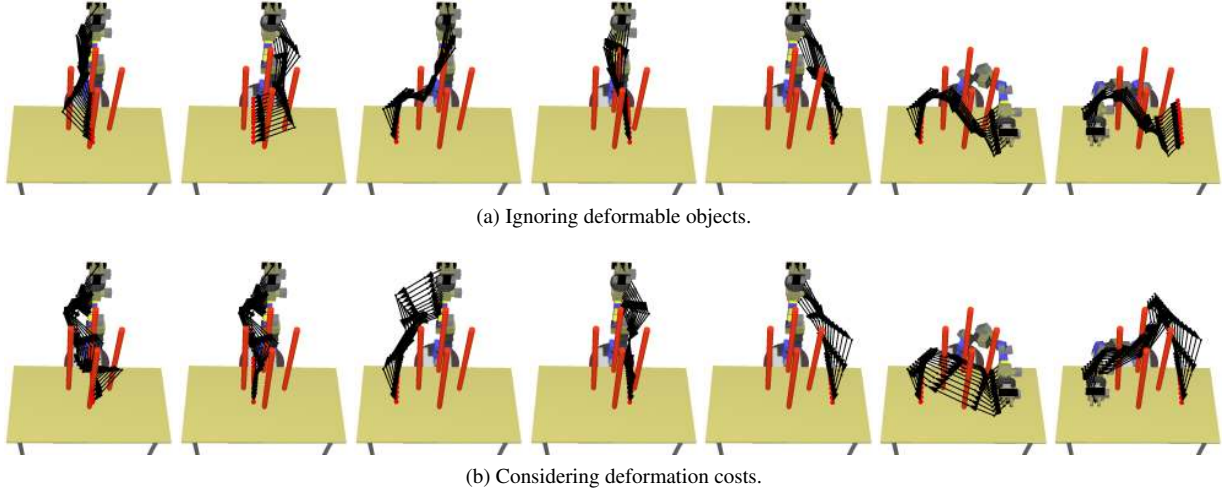


Figure 20: Comparison of seven planning tasks (T1-T7 in Table 3) in an environment with deformable rods for a planner that ignores deformable objects and our planner. Our planner chooses longer trajectories that lead to lower deformation costs.

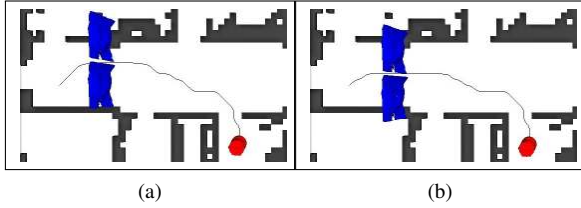


Figure 22: The planner prefers trajectories that minimize object deformations. The curtains in setup (a) are moved 40 cm along the positive y-axis compared to the setup from the previous experiment (b). The weighting coefficient α is set to 0.2 in both examples.

ing for a path, and a weighting coefficient α (see Eq. 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 21. 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 22 shows the generated tra-

jectory 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: We evaluated how well our sensor model for deformable objects is able to predict the presence of deformable objects during robot navigation. We determined sensor models for two different trajectories through both curtains with minimal deformation costs that were chosen preferably by our planner. 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 Table 4 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 %.

Recognition of dynamic obstacles: The sensor model is able to distinguish well between deformable and static non-deformable objects contained in the map of the robot. For collision avoidance, however, the key question is whether the system is able to distinguish well between deformable objects and close-by dynamic obstacles not contained in the map, given that the dynamic obstacles are not occluded by deformable objects and can be perceived by the sensor. Therefore, we performed

Planning task	deformation weight α	Path length	deformation costs
T1:	0.0	659.3	83.5
	0.5	1122.9	14.6
T2:	0.0	674.6	69.9
	0.5	988.3	4.7
T3:	0.0	539.4	515.6
	0.5	821.0	0.3
T4:	0.0	595.6	31.6
	0.5	854.5	19.7
T5:	0.0	668.2	45.3
	0.5	720.2	0.1
T6:	0.0	741.3	6718.7
	0.1	758.5	4346.1
T7:	0.0	741.3	3786.3
	0.1	1101.1	2736.5

Table 3: Planner evaluation in the rods environment: we compare the resulting deformation costs and path lengths of trajectories computed using our approach to those resulting from a planner that ignores deformable objects ($\alpha = 0.0$).

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 5. 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

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

Table 4: Confusion matrix for predicting, whether a sensor measurement corresponds to a deformable object in a static environment.

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

Table 5: Confusion matrix for predicting, whether a sensor measurement corresponds to a deformable object in an environment containing both deformable and dynamic objects.

region of correctly classified measurements observing a dynamic obstacle. Thus, the robot was still able to recognize dynamic obstacles and to avoid collisions with them.

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 23 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.⁵

Outdoor Robot Example: As a further application scenario for our planner, we consider a tractor-like outdoor robot that navigates in a tree plantation. We address this problem in simulation, since we do not have an outdoor robot with force-sensing capabilities. In this example, the tree trunk is assumed to be rigid while the treetop is modeled to be uniformly elastic. Given this setup, our planner is able to consider deformations of the twigs and chooses a trajectory with low deformation costs as illustrated in Figure 24.

7.3.3. 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 25. Both planners generally avoid the rubber ducks. Our planner

⁵Real-world experiment: http://www.informatik.uni-freiburg.de/~bfrank/videos/albert_curtains.avi



Figure 23: The mobile robot Albert is navigating through curtains.

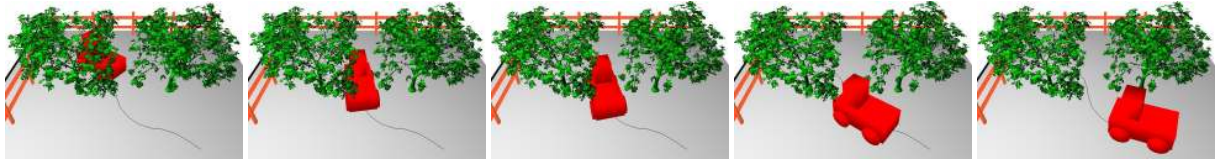


Figure 24: Application scenario for a simulated outdoor robot that navigates among vegetation. Our planner chooses a low-deformation cost trajectory that leads to gentle deformations of the low-hanging twigs of the tree.

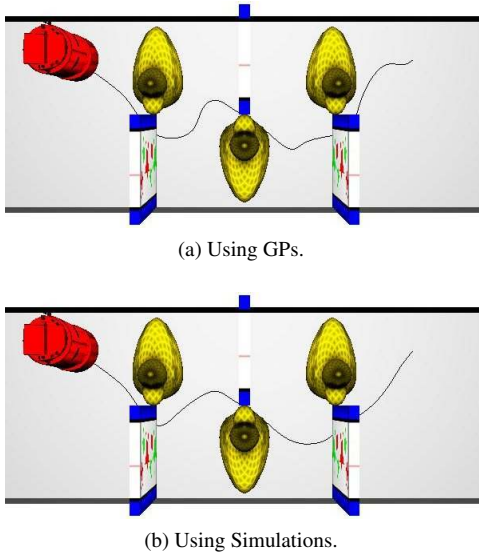


Figure 25: Comparison of our planning system (a) with a planning system that performs simulations during run-time (b).

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.

7.3.4. 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, a model for the static part of the world has to be determined. The occupancy grid map for the 2D real-world navigation example was determined independently and readily available with the CARMEN software package. The roadmap computation for the static part of the real-world 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 motions represented in the static model can be evaluated using our GP-based regression approach. In order to speed up planning, the deformation costs of edges in the roadmap that potentially lead to object deformations can be precomputed. This required 3 s for the 2D grid map and 420 s for the 3D roadmap in the foam environment. 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 be solved efficiently, even for manipulation robots with several degrees of freedom.

Comparison to a roadmap planner with integrated simulation: 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 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. When answering path queries, the initial and goal configuration would 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. 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. A further advantage of our deformation cost functions is that they describe the costs relative to an object and need to be computed only once for each type of object, while a roadmap precomputed as described above would have to be recomputed whenever the environment changes.

8. Conclusion

In this paper, we presented several techniques to enable robot motion in environments with deformable obstacles. 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.

Our robot is equipped with the sensors necessary to acquire models of deformable objects and determines their material parameters by minimizing the error between observed deformation and model prediction. 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 can be predicted accurately.

To realize an efficient planning system and to avoid time-consuming simulations during planning time, we introduced deformation cost functions for objects based on Gaussian process (GP) models. Our roadmap-based motion planner considers object deformations by optimizing the trade-off between motion costs and deformation costs. It determines the deformation costs of path segments in the roadmap using GP-based regression. 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. In several applications, 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

The authors would like to thank Axel Rottmann for sharing his expertise on Gaussian process regression and Jörg Müller for his help with robotic hardware issues. This work has partly been supported by the DFG under contract number SFB/TR-8, by the European Commission under FP7-248258-First-MM, and by Microsoft Research, Redmond.

References

- [1] U. Frese, G. Hirzinger, Simultaneous Localization and Mapping - A Discussion, in: Proc. of the IJCAI Workshop on Reasoning with Uncertainty in Robotics, 2001.
- [2] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, C. Hertzberg, Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.
- [3] V. Ila, J. Porta, J. Andrade-Cetto, Information-based Compact Pose SLAM, IEEE Transactions on Robotics 26 (1) (2010) 78–93.
- [4] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, F. Dellaert, iSAM2: Incremental Smoothing and Mapping with Fluid Relinearization and Incremental Variable Reordering, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2011.
- [5] B. Yamauchi, Frontier-Based Exploration Using Multiple Robots, in: Proc. of the Int. Conf. on Autonomous Agents, 47–53, 1998.
- [6] K. Wurm, C. Stachniss, W. Burgard, Coordinated Multi-Robot Exploration using a Segmentation of the Environment, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS), 2008.
- [7] P. Whaithe, F. P. Ferrie, Autonomous Exploration: Driven by Uncertainty, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (3) (1997) 193–205.

- [8] F. Bourgoult, A. Makarenko, S. Williams, B. Grocholsky, F. Durrant-Whyte, Information Based Adaptive Robotic Exploration, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS), Lausanne, Switzerland, 2002.
- [9] C. Stachniss, G. Grisetti, W. Burgard, Information Gain-based Exploration Using Rao-Blackwellized Particle Filters, in: Proc. of Robotics: Science and Systems (RSS), 2005.
- [10] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publ., 1991.
- [11] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, S. Thrun, Principles of Robot Motion, MIT Press, 2005.
- [12] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
- [13] E. Anshelevich, S. Owens, F. Lamiroux, L. E. Kavraki, Deformable Volumes in Path Planning Applications, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2000.
- [14] O. B. Bayazit, J.-M. Lien, N. M. Amato, Probabilistic Roadmap Motion Planning for Deformable Objects, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2002.
- [15] R. Gayle, P. Segars, M. Lin, D. Manocha, Path Planning for Deformable Robots in Complex Environments, in: Proc. of Robotics: Science and Systems (RSS), 2005.
- [16] R. Alterovitz, K. Goldberg, J. Pouliot, I. Hsu, Sensorless Motion Planning for Medical Needle Insertion in Deformable Tissues, IEEE Transactions on Information Technology in Biomedicine 13 (2) (2009) 217–225.
- [17] B. Maris, D. Botturi, 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.
- [18] S. Rodríguez, J.-M. Lien, N. M. Amato, Planning Motion in Completely Deformable Environments, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2006.
- [19] S. Patil, J. van den Berg, R. Alterovitz, Motion Planning Under Uncertainty In Highly Deformable Environments, in: Proc. of Robotics: Science and Systems (RSS), 2011.
- [20] M. Metzger, M. Gissler, M. Asal, M. Teschner, Simultaneous Cutting of Coupled Tetrahedral and Triangulated Meshes and its Application in Orbital Reconstruction, Int. Journal of Computer Assisted Radiology and Surgery 4 (5) (2009) 409–416.
- [21] D. Chen, D. Zeltzer, Pump it up: Computer animation of a Biomechanically Based Model of Muscle using the Finite Element Method, in: Proceedings of ACM SIGGRAPH, 1992.
- [22] G. Picinbono, H. Delingette, N. Ayache, Non-linear and anisotropic elastic soft tissue models for medical simulation, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2001.
- [23] M. Teschner, B. Heidelberger, M. Müller, M. Gross, A Versatile and Robust Model for Geometrically Complex Deformable Solids, in: Proc. of Computer Graphics International, 2004.
- [24] F. Conti, O. Khatib, 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.
- [25] A. Nealen, M. Müller, R. Keiser, E. Boxerman, M. Carlson, Physically Based Deformable Models in Computer Graphics, Computer Graphics Forum 25 (4) (2006) 809–836.
- [26] K.-J. Bathe, Finite Element Procedures, Prentice Hall, 2nd edn., 1996.
- [27] M. Hauth, W. Strasser, Corotational Simulation of Deformable Solids, in: Int. Conf. on Computer Graphics, Visualization, and Computer Vision (WSCG), 2004.
- [28] M. Müller, M. Gross, Interactive Virtual Materials, in: Graphics Interface, 2004.
- [29] S. R. Mousavi, I. Khalaji, A. S. Naini, K. Raahemifar, A. Samani, Statistical finite element method for real-time tissue mechanics analysis, Computer Methods in Biomechanics and Biomedical Engineering 15 (6) (2012) 595–608.
- [30] K. K. Hauser, C. Shen, J. F. O'Brien, Interactive Deformation Using Modal Analysis with Constraints, Graphics Interface (2003) 247–256.
- [31] M. G. Choi, H.-S. Ko, Modal Warping: Real-Time Simulation of Large Rotational Deformation and Manipulation, IEEE Transactions on Visualization and Computer Graphics 11 (1) (2005) 91–101.
- [32] J. Kim, N. S. Pollard, Fast Simulation of Skeleton-Driven Deformable Body Characters, ACM Transactions on Graphics 30 (5) (2011) 121:1–121:19.
- [33] J. Barbič, D. L. James, Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models, ACM Transactions on Graphics (ACM SIGGRAPH) 24 (3) (2005) 982–990.
- [34] G. Bianchi, B. Solenthaler, G. Székely, 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.
- [35] B. Lloyd, G. Székely, M. Harders, Identification of Spring Parameters for Deformable Object Simulation, IEEE Trans. on Visualization and Computer Graphics 13 (5) (2007) 1081–1094.
- [36] P. Fong, Sensing, Acquisition, and Interactive Playback of Data-based Models for Elastic Deformable Objects, Int. Journal of Robotics Research 28 (5) (2009) 630–655.
- [37] B. Bickel, M. Baecher, M. Otaduy, W. Matusik, H. Pfister, M. Gross, Capture and Modeling of Non-Linear Heterogeneous Soft Tissue, Proc. of ACM SIGGRAPH 28 (3) (2009) 1081–1094.
- [38] J. Kajberg, 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) (2004) 3439–3459.
- [39] A. P. C. Choi, 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) (2005) 258–264.
- [40] D. S. Schnur, N. Zabarar, An Inverse Method for Determining Elastic Material Properties and a Material Interface, Int. Journal for Numerical Methods in Engineering 33 (10) (1992) 2039–2057.
- [41] M. Becker, M. Teschner, Robust and Efficient Estimation of Elasticity Parameters Using the Linear Finite Element Method, in: Proc. of Simulation and Visualization, 2007.
- [42] M. Kauer, V. Vuskovic, J. Dual, G. Székely, M. Bajka, Inverse Finite Element Characterization of Soft Tissues, Medical Image Analysis 6 (3) (2002) 275–287.
- [43] A. R. Fugl, A. Jordt, H. G. Petersen, M. Willatzen, 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.
- [44] J. Lang, D. K. Pai, R. J. Woodham, Acquisition of Elastic Models for Interactive Simulation, Int. Journal of Robotics Research 21 (8) (2002) 713–733.
- [45] P. Boonvisut, R. Jackson, 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.
- [46] J. Schulman, A. Lee, J. Ho, P. Abbeel, Tracking Deformable Objects with Point Clouds, in: Proc. of the IEEE Int. Conf. on

- Robotics & Automation (ICRA), 2013.
- [47] C. Holleman, L. E. Kavraki, J. Warren, Planning Paths for a Flexible Surface Patch, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 1998.
 - [48] A. Mahoney, J. Bross, D. Johnson, Deformable Robot Motion Planning in a Reduced-Dimension Configuration Space, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.
 - [49] A. Jain, M. D. Killpack, A. Edsinger, C. C. Kemp, Reaching in clutter with whole-arm tactile sensing, *Int. Journal of Robotics Research* 32 (4) (2013) 458–482.
 - [50] C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, 2006.
 - [51] T. Lang, C. Plagemann, W. Burgard, Adaptive Non-Stationary Kernel Regression for Terrain Modeling, in: Proc. of Robotics: Science and Systems (RSS), 2007.
 - [52] S. Vasudevan, F. T. Ramos, E. W. Nettleton, H. F. Durrant-Whyte, Gaussian Process Modeling of Large Scale Terrain, *Journal of Field Robotics* 26 (10) (2009) 812–840.
 - [53] S. O’Callaghan, F. T. Ramos, H. F. Durrant-Whyte, Contextual Occupancy Maps incorporating Sensor and Location Uncertainty, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.
 - [54] J. Ko, D. Fox., GP-BayesFilters: Bayesian Filtering Using Gaussian Process Prediction and Observation Models, *Autonomous Robots* 27 (1) (2009) 75–90.
 - [55] C. Stachniss, C. Plagemann, A. J. Lilienthal, Gas Distribution Modeling using Sparse Gaussian Process Mixtures, *Autonomous Robots* 26 (2-3) (2009) 187–202.
 - [56] P. Henry, C. Vollmer, B. Ferris, D. Fox, Learning to Navigate Through Crowded Environments, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.
 - [57] L. Murphy, P. Newman, Planning Most Likely Paths from Overhead Imagery, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2010.
 - [58] B. Frank, M. Becker, C. Stachniss, M. Teschner, W. Burgard, Efficient Path Planning for Mobile Robots in Environments with Deformable Objects, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2008.
 - [59] B. Frank, C. Stachniss, R. Schmedding, M. Teschner, W. Burgard, Real-world Robot Navigation amongst Deformable Obstacles, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2009.
 - [60] B. Frank, R. Schmedding, C. Stachniss, M. Teschner, 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), 2010.
 - [61] B. Frank, C. Stachniss, N. Abdo, 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), 2011.
 - [62] L. E. Kavraki, P. Svestka, J.-C. Latombe, M. H. Overmars, Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces, *IEEE Transactions on Robotics and Automation* 12 (4) (1996) 566–580.
 - [63] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, M. Gross, Optimized Spatial Hashing for Collision Detection of Deformable Objects, in: Proc. Vision, Modeling, Visualization (VMV), 2003.
 - [64] J. Spillmann, M. Becker, M. Teschner, Non-Iterative Computation of Contact Forces for Deformable Objects, *Journal of Computer Graphics, Visualization, and Computer Vision (WSCG)* 15 (1–3) (2007) 33–40.
 - [65] T. J. Chung, *Applied Continuum Mechanics*, Cambridge University Press, New York, 1996.
 - [66] M. Hauth, W. Strasser, Corotational Simulation of Deformable Solids, *Journal of WSCG* 12 (1–3) (2004) 137–145.
 - [67] P. J. Besl, N. D. McKay, A Method for Registration of 3-D Shapes, *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 14 (2) (1992) 239–256.
 - [68] K. Pulli, Multiview Registration for Large Data Sets, in: Proc. of the Int. Conf. on 3D Digital Imaging and Modeling (3DIM), 1999.
 - [69] S. Rusinkiewicz, M. Levoy, Efficient Variants of the ICP Algorithm, in: Proc. of the Int. Conf. on 3D Digital Imaging and Modeling (3DIM), 2001.
 - [70] B. K. P. Horn, Closed-Form Solution of Absolute Orientation Using Unit Quaternions, *Journal of the Optical Society A* 4 (4) (1987) 629–642.
 - [71] J. Spillmann, M. Wagner, M. Teschner, Robust Tetrahedral Meshing of Triangle Soups, in: Proc. Vision, Modeling, Visualization (VMV), 2006.
 - [72] M. Riedmiller, 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.
 - [73] R. M. Neal, *Bayesian Learning for Neural Networks*, Lecture Notes in Statistics No. 118, Springer, 1996.
 - [74] C. K. I. Williams, *Computation with Infinite Neural Networks*, *Neural Computation* 10 (5) (1998) 1203–1216.
 - [75] C. K. I. Williams, Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond, in: M. I. Jordan (Ed.), *Learning in Graphical Models*, The MIT Press, 599–621, 1999.
 - [76] M. Montemerlo, N. Roy, S. Thrun, D. Haehnel, C. Stachniss, J. Glover, *Carmen – Robot Navigation Toolkit*, <http://carmen.sourceforge.net/home.html>, 2008.
 - [77] D. Fox, W. Burgard, S. Thrun, The Dynamic Window Approach to Collision Avoidance, *IEEE Robotics & Automation Magazine* 4 (1).
 - [78] J. Minguez, 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.