Enabling Motion Planning and Execution for Tasks Involving Deformation and Uncertainty

by

CALDER PHILLIPS-GRAFFLIN

A Dissertation Submitted to the Faculty of the
Worcester Polytechnic Institute
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in
Robotics Engineering

June 2017

Approved as to style and content by:

Dmitry Berenson, Advisor

Cagdas Onal, Member

Raghvendra Cowlagi, Member

Alberto Rodriguez, Member

Abstract

A number of outstanding problems in robotic motion and manipulation involve tasks where degrees of freedom (DoF), be they part of the robot, an object being manipulated, or the surrounding environment, cannot be accurately controlled by the actuators of the robot alone. Rather, they are also controlled by physical properties or interactions – contact, robot dynamics, actuator behavior – that are *influenced* by the actuators of the robot. In particular, we focus on two important areas of poorly controlled robotic manipulation: motion planning for deformable objects and in deformable environments; and manipulation with uncertainty. Many everyday tasks we wish robots to perform, such as cooking and cleaning, require the robot to manipulate deformable objects. The limitations of real robotic actuators and sensors result in uncertainty that we must address to reliably perform fine manipulation. Notably, both areas share a common principle: contact, which is usually prohibited in motion planners, is not only sometimes unavoidable, but often necessary to accurately complete the task at hand.

We make four contributions that enable robot manipulation in these poorly controlled tasks: First, an efficient discretized representation of elastic deformable objects and cost function that assess a "cost of deformation" for a specific configuration of a deformable object that enables deformable object manipulation tasks to be performed without physical simulation. Second, a method using active learning and inverse-optimal control to build these discretized representations from expert demonstrations. Third, a motion planner and policy-based execution approach to manipulation with uncertainty which incorporates contact with the environment and compliance of the robot to generate motion policies which are then adapted during execution to reflect actual robot behavior. Fourth, work towards the development of an efficient path quality metric for paths executed with actuation uncertainty that can be used inside a motion planner or trajectory optimizer.

Acknowledgements

I would like to thank my adviser, Dmitry Berenson, for his support, encouragement, and guidance throughout my graduate studies. My committee, Cagas Onal, Raghvendra Cowlagi, and Alberto Rodriguez, have provided thoughtful feedback and helpful suggestions. My fellow lab members and graduate students have provided endless hours of helpful conversation and assistance with robots and experiments. Last but not least, I want to thank my friends and family for their support, patience, and hospitality, particularly during my final experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many outstanding problems in robotic motion and manipulation involve tasks where degrees of freedom (DoF) cannot be accurately controlled. These DoF, be they part of the robot, an object being manipulated, or the surrounding environment, cannot be accurately controlled by the actuators of the robot alone. Rather, they are also controlled by physical properties or interactions – contact, robot dynamics, actuator behavior – that are *influenced* by the actuators of the robot. Even though they cannot be controlled directly, these DoF must be accounted for to enable robust robot behavior. Some robots, such as many popular robotic hands, deliberately include underactuated mechanisms in their design to reduce the number of actuators. Instead, we focus on difficult-to-control systems that arise indirectly as a result of the task being performed or undesired behavior of the robot.

In particular, we focus on two important areas of poorly controlled robotic manipulation: motion planning for deformable objects and in deformable environments; and manipulation with uncertainty. Many everyday tasks we wish robots to perform, such as cooking and cleaning, require the robot to manipulate deformable objects. Other important applications of deformable object manipulation include industrial and surgical tasks. Here, the high number of DoF that arise from the deformable materials cannot be controlled directly, but instead are influenced primarily by gravity and contact. The limitations of real robotic actuators and sensors result in uncertainty that we must address to reliably perform fine manipulation. For example, the widespread use of safer compliant and low-cost robot manipulators including Baxter and Raven has led to a class of robots which exhibit significant actuator error yet also have accurate sensors. Here, the poorly-controlled DoF arise from the uncertainty of the actuators, which can be influenced by the dynamics of the robot and contact with the environment. Notably, both areas share a common principle: contact, which is usually prohibited in motion planners, is not only sometimes unavoidable, but often necessary to accurately complete the task at hand.

This thesis makes four contributions that enable robot manipulation in these poorly controlled tasks: First, an efficient discretized representation of elastic

deformable objects and cost function that assess a "cost of deformation" for a specific configuration of a deformable object that enables deformable object manipulation tasks to be performed without physical simulation. Second, a method using active learning and inverse-optimal control to build these discretized representations from expert demonstrations. Third, a motion planner and policy-based execution approach to manipulation with uncertainty which incorporates contact with the environment and compliance of the robot to generate motion policies which are then adapted during execution to reflect actual robot behavior. Fourth, work towards the development of an efficient path quality metric for paths executed with actuation uncertainty that can be used inside a motion planner or trajectory optimizer.

We seek to plan motion for deformable robots and environments that minimizes deformation. Previous work in motion planning for deformable robots and environments has relied on the use of expensive Mass-Spring (M-S) [1] and Finite Element (FEM) models [2] of the deformable elements. While these models can produce accurate models of deformed geometry, they are too expensive to use inside a motion planner that must evaluate thousands of possible configurations. Instead, inspired by work on efficient meshless models [3], we introduce a discretized representation of deformable objects. Importantly, this representation contains parameters for both quantitative physical *properties* - i.e., the stiffness of the material, but also quantitative *characteristics* - the "sensitivity" of the object. This distinction allows us to encode not only material properties, but properties like fragility or importance, which allows a motion planner to increase deformation of other objects to avoid deforming a particularly sensitive one.

Using these two sets of parameters, we develop a cost function that computes a "cost of deformation" for colliding (i.e., deforming) configurations and can be used in standard optimal and cost-sensitive motion planners such as A*, RRT*, T-RRT, and GradienT-RRT. Further development of our representation introduced an efficient online method for detecting motions that puncture a deformable object. Such motions, while often low cost, are highly undesirable and must be explicitly prohibited. Using our representation and cost function, we have planned low- ($SE(2)$) and high-dimensional ($SE(3)$) paths for rigid and deformable objects in deformable or rigid environments without requiring expensive physical simulation of deformable object behavior.

While our "cost of deformation" and discretized models enable efficient motion planning without needing expensive physical simulation, the qualitative *sensitivity* parameter introduces a critical new parameter to tune. As a parameter that captures *qualitative* characteristics of an object, it often cannot be set directly from known physical properties. This is particularly important in complex environments, in which accurate object sensitivities are critical to producing desirable motion. Indeed, it is often easier to specify the desired motion than the parameters that produce it. Instead of time-consuming and error-prone manual parameter tuning, we frame the problem as one of inverse-optimal control (IOC), in which we use optimal expert demonstrations to recover the desired sensitivity parameters. IOC has been widely used in robot control

and learning [4, 5, 6, 7], and offers the ability to use domain knowledge of the task (for example, a surgeon familiar with the tissues and organs of the body) to produce parameters for complex and unintuitive cost functions.

Our approach uses the *Path Integral Inverse Reinforcement Learning* (PI-IRL) [8] IOC technique, which uses optimal demonstration trajectories and a set of randomly-sampled suboptimal trajectories to compute optimal parameter values using convex optimization. Notably, using PIIRL avoids the need to repeatedly solve the forward problem – in this case, the forward problem consists of planning an optimal minimum-deformation path using the asymptotically-optimal RRT* [9] planner, an operation which takes approximately 30 minutes. PIIRL, of course, requires that sufficient demonstrations be collected to learn the desired behavior. Too few collected, and incorrect behavior will result, while exhaustively collecting all possible demonstrations (for example, a demonstration for every pair of deformable objects in an environment) would be time-consuming and potentially infeasible. To address this, we introduce an active learning algorithm that selects demonstrations needed to cover all objects in the environment. Using the active learning and PIIRL, we accurately recover sensitivity parameters for deformable objects in a simulated surgical probe insertion task and show that planning using the recovered parameters reproduces the expert-demonstrated motion.

Our contribution to manipulation with actuation uncertainty incorporates contact with the environment and compliance of the robot to generate motion policies which we then adapt during execution to reflect actual robot behavior. This approach consists of two steps: First, an anytime sampling-based motion planner which generates policies containing multiple solution paths; and Second, policy-based execution which compares the real configurations reached by the robot to those predicted by the planner and updates the policy accordingly. Not only is contact with the environment often unavoidable due to the robot's actuator uncertainty, but it can be beneficial, as contact reduces uncertainty. Our planner uses forward-simulation of the robot's motion and compliance when in contact with the environment to predict the configurations that result from specified control inputs. While some previous approaches have used analytical probability distributions to model the distribution of potential configurations of a robot with uncertainty (what we call *belief*), contact with the environment results in some dimensions of the belief distribution losing support or the belief becoming trans-dimensional, which these analytical distributions cannot model. Instead, like previous work [10], we use a particle-based representation of uncertainty in which we forward-simulate multiple noisy motions of the robot for each control input.

Similarly to previous work [10], we note that multiple attempts to perform a single *action* may result in multiple *outcomes*. While we cannot select between these outcomes, we can be resilient to undesirable behavior by identifying during execution which outcome is reached, and if need be, attempt to return to the previous state and try again. We incorporate this resilience into execution using policies that not only detect when undesirable outcomes are reached, but also unexpected outcomes that were not encountered during planning. We use

the robot behavior during execution to update the policy so that the policy reflects the true ability of the robot to perform actions; for example, if an action is in fact impossible to complete due to an obstruction or unmodelled dynamics of the robot. In our experiments in a range of low- ($SE(2)$) and high-dimensional ($SE(3)$ and R^7) manipulation tasks, our approach generates policies that reliably perform manipulation tasks involving significant contact and are resilient to significant changes during execution.

Our work in motion planning and execution with actuation uncertainty offers a method for reliably performing fine manipulation tasks; however, it is still significantly more computationally expensive than simpler sampling-based techniques. While these simpler techniques do not account for uncertainty (or even simpler techniques that forbid contact entirely), our experience shows that they can accomplish some tasks with similar probability of success (see Section 5.4.2 for such an example) so long as the planned policy is robust and diverse. Indeed, because simpler approaches that ignore uncertainty require significantly less simulation, they produce policies with more diverse solution paths. Likewise, a conventional planner that did not require *any* simulation would likely produce even more diverse policies. However, all of these simplified approaches lose the ability to assess the quality of the paths they generate.

We wish to develop a metric to assess the quality of these paths; in particular, we seek a metric that can be used during the motion planning process. Such a metric has a number of important applications: by independently assessing path quality, we can use simpler, more efficient, planners without losing guarantees of path quality; it can be used to distinguish between tasks which can be solved with simpler planners and those that require planning incorporating uncertainty; or it can be used to assess the quality of planned paths in the face of potential changes to robot's actuation uncertainty or the environment. Significantly, the availability of an efficient path quality metric enables the use of trajectory optimization techniques. While we can empirically assess the quality of these planned paths using simulation, forward-simulating the large number of executions necessary to accurately measure path quality is prohibitively expensive.

Instead, we propose a metric using the reachable $\mathcal{C}$-space volume of the path. We construct a set of reachable $\mathcal{C}$-space volumes between the waypoints of the path that bound the possible configurations the robot could achieve given the actuation uncertainty. Within each reachable volume, we can determine which configurations become stuck and fail to reach the target waypoint, which we use to characterize stuck regions of the contact manifold. We then estimate the probability that samples reach these stuck regions. We combine these probabilities for all reachable volumes along a path to compute the probability of the robot becoming stuck, and thus form a metric for path quality.

In this thesis, we start by presenting the representation of deformable objects and cost function we developed to enable motion planning for deformable robots and environments without expensive simulation in Chapter 3. Further developing our representation and motion planning, in Chapter 4 we present our active learning and IOC-based approach for learning behavior in deformable en-

vironments from expert demonstration. Our development of a motion planner and policy-based execution method for robots with actuation uncertainty is presented in Chapter 5. Our work on motion planning for robots with actuation uncertainty motivates the development of a path quality metric for paths planned for these robots that can be used within the planning process. In Chapter 6, we formally define the problem of computing a path quality metric for execution under actuation uncertainty and develop a method for efficiently computing such a metric.

Chapter 2

Related Work

2.1 Modeling deformable objects

Deformable objects are important for a variety of fields ranging from computer graphics to robotics to medicine. As a result, extensive work has been done on the modelling and representation of deformable objects for computer graphics [11] and medicine [12]. More recently, this work has been adapted to robotics to allow the manipulation of real-world objects such as clothing, rope, and human tissue. Unlike a range of work focusing on visual servoing [13, 14], realtime simulation [15, 16], haptics [17, 18], and learning from demonstration [19, 20] with deformable objects, we focus on motion planning for deformable objects – i.e., we seek to compute a path that minimizes deformation. [16, 21]

While a wide variety of modelling approaches for deformable objects exist for computer graphics and physical simulation purposes, we are primarily concerned with representations suitable for motion planning purposes. We wish to avoid the problem of directly computing the geometry of a deformed object, as doing so is an expensive intermediate step to assessing the severity of deformation.

Existing representations of deformable objects fall into two main groups, those using meshed volumetric models and those using meshless models:

Meshed - Often tetrahedral meshes, these models preserve volumetric constraints and allow the use of numerical simulation to compute the effects of collision. Existing work uses Mass-Spring (M-S) [11] and Finite-Element (FEM) [22],[23] methods to simulate changes to object geometry resulting from collisions. These methods allow the inclusion of volume preservation and restoration. However, as noted in [1], M-S models are inaccurate beyond low-deformation cases, and both M-S and FEM are expensive to compute.

Meshless - Two variants of this approach exist – models that represent only the surface of the object such as [1], and models such as [3], which use a discretized representation that allows for heterogeneous objects with varying internal properties. In the former case, the surface model was used to compute the penetration into the object, ignoring internal forces. In the latter, discrete

information on material properties allows the efficient computation of object deformation with comparable accuracy to established FEM methods [3].

Some deformable objects, such as rope and thread, can change shape with limited (or no) restoring forces. Existing models for these objects assume that the object itself is incompressible [24] (i.e.,, a rope cannot be crushed), which prevents the application of these techniques to problems such as ours, in which the objects modelled must be compressible. Other models for these objects, such as [25, 21] use approximate Jacobian-based models for online modelling and control, but are unsuitable for our global planning approach.

The most similar work to ours is [3] which uses underlying discrete representation similar to ours, [3] simulates the geometry and kinematics of heterogeneous deformable objects. While this technique could be used as a stepping-stone to compute a cost of deformation, our method skips the simulation step to directly compute a cost value. In addition, this approach does not incorporate an element comparable to the *sensitivity* used in our representation.

Learning physical parameters of deformable objects from demonstration and manipulation has been explored using camera and pointcloud observations [26, 27, 28], and through active manipulation [29, 15, 30]. Limited work has been done using optimization to learn parameters; in [27] parameters of an FEM model were learned by iteratively updating model parameters until modelled object deformation and interaction forces matched those measured when deforming a test object. However, these approaches are limited to capturing the observable physical behavior of a given object.

2.2 Planning for deformable objects

Building from these established representations and techniques, a range of motion planning approaches have been developed to find paths in deformable environments. Extensive work has been done applying Probabilistic Roadmap planners (PRM) [31] and Rapidly-exploring Random Tree planners (RRT) [32] to deformable objects, including [33, 34, 35, 36, 37, 2]. Other work in the area includes planning for rope and thread such as [24].

However, the above work is marked by a trade-off between the desires for accuracy and performance. Accurate methods based on FEM models are slow to compute, prompting a range of simpler models such as [34, 1] which are designed to provide "good-enough" simulation of deformation. Furthermore, most previous work is concentrated on finding *feasible* deformations using volume preservation and penetration distance to evaluate feasibility, whereas we seek to minimize deformation.

Limited work has been done to account for the severity of the deformation, such as [1] and [2], which assess a cost of deformation. In the former, the problem of motion planning itself is replaced with an optimization problem of reducing the cost for motion along a parametrized Bezier curve below a preset threshold. Instead of a local optimization approach like this, our approach is to use global planners.

The most similar planning approach to ours, [2], attempts to generate optimal paths, taking into account both costs incurred in deformation and path length. Unlike our approach, [2] relies on extensive and time-consuming precomputation of a meshed environment using FEM methods. Notably, because our approach provides a cost function that accounts for both object deformation and sensitivity, it produces plans that both minimize deformation and preferentially deform or avoid objects based on their sensitivity.

2.3 Inverse optimal control (IOC)

Inverse Optimal Control (IOC) is the problem of recovering the cost or reward function being optimized by a trajectory or policy. Introduced by Kalman [4] and applied to a range of robotics problems [5, 38, 39], several different formulations of the IOC problem and algorithms to address it have been proposed, covering both continuous and discrete state spaces [5]. Earlier approaches to the IOC problem, such as apprenticeship learning, require that the forward problem be solved in addition to computing optimal weights [7, 6]. More recent approaches, based on the maximum entropy principle, replace the need for solving the forward problem by using sample trajectories around the demonstration [40].

The IOC approach we use, *Path Integral Inverse Reinforcement Learning* (PIIRL) samples around the demonstration instead of solving the forward problem [41, 8]. In the PIIRL formulation, a series of locally-optimal demonstration trajectories are gathered from the user(s). For each of these demonstrations, a set of sample trajectories around the demonstration is generated; note that these samples are assumed to be sub-optimal relative to their demonstration. For all demonstrations and all samples, user-specified features are evaluated, and the weights associated with these features are then recovered via convex optimization that attempts to maximize the margin between the features of the demonstrations and the features of the samples.

2.4 Planning and manipulation with uncertainty

Planning motion in the presence of actuation uncertainty dates back to the seminal work of Lozano-Pérez et al. [42], which introduced pre-image backchaining. A pre-image, i.e., a region of configuration space from which a motion command attains a certain goal recognizably, was used in a planner that produced actions guaranteed to succeed despite pose and action uncertainty. However, constructing such pre-images is prohibitively computationally expensive [43, 44, 45].

In its general form, belief-space planning is formulated as a Partially-Observable Markov Decision Process (POMDP) [46], which are widely known to be intractable for high-dimensional problems [47, 44]. However, recent developments of online planners [48] and general approximate point-based solvers such as SARSOP [49], MCVI [50], DESPOT [51], and others [52, 53] have made consid-

erable progress in generating policies for complex POMDP problems. For some lower-dimensional robot motion problems like [54, 55, 56], the POMDP can be simplified by extracting the part of the task that incorporates uncertainty (e.g., the position of an item to be grasped) and applying off-the-shelf solvers to the problem. Online approaches also exist, such as [51, 57]. In certain cases, MDP and POMDP problems can be linearized [58, 59]. Others have investigated learning approaches [60] for similar problems; however, we are interested in planning because we want our methods to generalize to a broad range of tasks without collecting new training data.

Recent work incorporates advances from sampling-based planning to plan motions in belief-space. This work uses derivatives of both PRM [61] and RRT [62] planners, including asymptotically-optimal derivatives such as [63, 64] which may produce nearly-optimal motion plans in certain combinations of robots and cost functions. Several sampling-based belief-space planners have been developed [10, 65, 66, 58, 67]. Others have evaluated sampling [68] and belief-space distance functions [69] and show that the selection of distance function greatly impacts the performance of the planner. Additionally, approaches using LQG and LQR controllers [58, 70, 71, 72, 73] and trajectory optimizers [74, 75] have been proposed. These approaches use Gaussian distributions to model uncertainty, but such a simple distribution cannot accurately represent the belief of a robot moving in contact with obstacles, where belief may lose support in one or more dimensions, or the state may become trans-dimensional. Other approaches like [10] use a set of particles to model belief like a particle filter; while we also use a particle-based representation, our approach more accurately captures the behavior of splits and also includes resilience during execution.

The importance of contact and compliance has long been known, with [42] demonstrating the important role of compliance in performing precise motion tasks. Others have investigated the effects of uncertainty on the use of contact in optimal control [76]. Sampling-based motion planning for compliant robots has been previously explored in [77], albeit limited to disc robots with simplified contact behavior. We draw from these methods, but our approach differs significantly from previous work by incorporating contact and compliance directly into the planning process by using forward simulation like the kinodynamic RRT [62]. A major advantage over existing methods is that the policies we generate are not fixed; instead, we update them online during execution, which allows us to reduce the impact of differences between our planning models and real-world execution conditions.

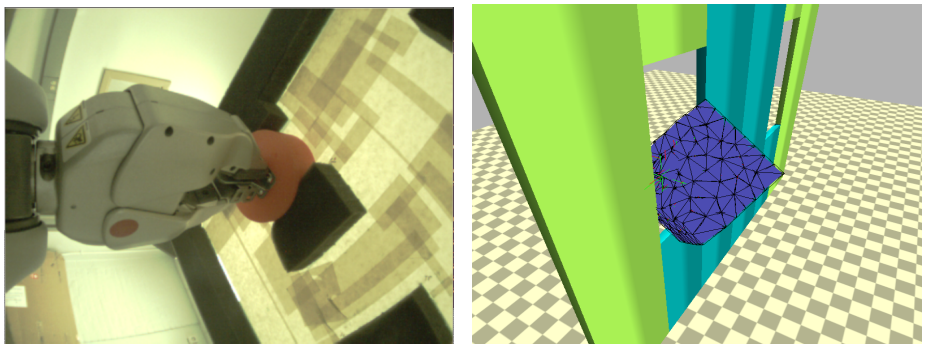
Chapter 3

Representation Of Deformable Objects For Motion Planning Without Physical Simulation

3.1 Introduction

The primary challenge of motion planning for deformable objects and environments is that deformation is very difficult to simulate accurately. Unlike rigid objects, whose dynamics are well-understood, the motion of a deformable object depends on a large and complex set of parameters that define its stiffness, friction, and volume preservation. Computing the geometry of a deformable object in contact with another object is particularly challenging, especially if both objects are deformable. Many methods exist for deformable object simulation, including mass-spring model simulation [11] and the more general finite element method (FEM) [22, 23]. Simulation methods for meshless models also exist [3]. FEM simulation is generally regarded to be the most accurate, although it is highly sensitive to the discretization of the deformable object and, for fine discretizations, is very time-consuming to compute.

Given the difficulties of deformable object simulation, we seek to explore the practicality of performing useful motion planning for deformable objects without explicitly simulating them. To accomplish this, we model the environment and moving object using voxel grids. Each contains two values: *deformability* and *sensitivity*. The deformability represents how compressible the voxel is. The sensitivity represents the penalty for deforming that particular voxel. Sensitivity is used to allow our motion planner to avoid some objects more than others in the planning process, which is useful if different objects have different sensitivity to deformation. For instance, in a surgical setting, one organ may be more sensitive



(a) (b)

Figure 3.1: Execution of paths planned using our representation: (a) with a PR2, (b) in the Bullet physics simulator.

to compression than another, even though both are equally deformable.

Our representation is designed to represent objects that deform elastically, meaning that while the object may change its surface geometry when in contact, it exhibits volume restoration and will return to its original shape when the colliding object is removed (e.g., a sponge). Note that our representation currently assumes that environment objects, whether deformable or rigid, do not move as a result of deformation or other forces. While restrictive, we believe that these limitations are consistent with a range of real-world problems where deformable objects are constrained by the environment (e.g., inside the body) or by the robot itself (e.g., a robot with deformable components). In cases that do not completely meet these constraints, we believe our representation will be conservative – it will over-estimate the severity of deformation. Certain objects such as clothing or rope that do not obey these limitations may instead be represented as an articulated series of these voxel-based representations, though this is not within the scope our work.

Once the moving object and environment have been defined using our voxel-based representation, we can evaluate the cost of a given configuration of the object by computing a novel cost function that combines deformability and sensitivity into a single value. This value represents the *deformation cost* of that configuration. Using this cost function, the cost of a given configuration in a motion planner may be computed. As we demonstrate, this cost function is suitable for both discrete (demonstrated using a variant of A*) and sampling-based motion planning algorithms (demonstrated using T-RRT [78] and GradienT-RRT [79]).

In our experiments we show that our method is effective at finding paths for rigid objects moving in deformable environments, deformable objects moving in rigid environments, and deformable objects moving in deformable environments. Computing paths for these scenarios would involve radically different simulation methods, however, using our approach, we need only to adjust the deformability

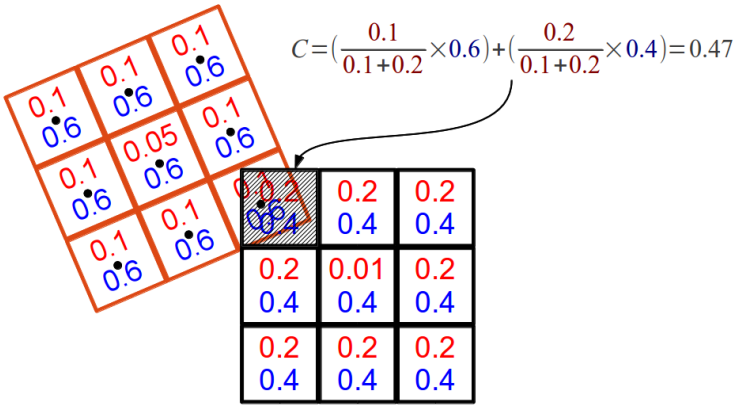


Figure 3.2: An example of our representation and cost function – deformability values are shown in red, sensitivity values are shown in blue. The voxel centers of the moving object (orange) are shown as black points. Cost is computed for the shaded voxel in collision.

of the object and environment. We verify the efficacy of our method in virtual environments using the Bullet physics simulator, and in physical experiments using the PR2 robot with a custom deformation-tracking camera system.

3.2 Methods

We have developed an efficient representation for deformable objects and a cost function for assessing the cost of collisions. Building from this representation, we have developed a cost function that allows discrete and sampling-based planners to compute paths that minimize deformation.

3.2.1 Representation

Object geometry is inherently captured in our voxel-based representation. Unlike triangle meshes and other methods optimized for surface representation, this discretization preserves information about the interior of the object. As with all discrete representations, the resolution of our representation is limited by the size on an individual voxel. Arbitrarily high resolution can be achieved by increasing the number and decreasing the size of the voxels, at the cost of increased memory usage and processing time. To address the well-known problem of rotating voxels, only the planning environment is directly modelled using voxels; objects being moved are represented by a set of points that shares the same discretization, in which the points correspond to voxel centers. An example representation is shown in Figure 3.2.

Our representation captures physical properties through the use of two parameters per voxel, *Sensitivity* and *Deformability*. These parameters represent

the cost incurred by deforming the voxel, and the ability of the voxel to be deformed (similar to the "stiffness" in [3]). Intuitively, a completely rigid object has a deformability of zero, while empty space has a deformability of one. Sensitivity is a user-assigned parameter that allows a range of object qualities to be represented. In general, sensitivity may be used to differentiate between two deformable objects with similar physical *properties* but different desired *qualities*.

Increasing sensitivities from the surface to the center of an object can be used to represent a greater severity of deformation (e.g., the tissue inside an organ may be much more prone to damage than the exterior) or, by increasing sensitivity to infinity, effectively prevent the planner from producing paths that result in any penetration. While deformability parameters can be derived from physical properties of an object, tuning sensitivity parameters is more complicated. In future work, we plan to investigate the automatic generation of sensitivity parameters.

3.2.2 Cost Function

Using our voxel-based representation, we develop a cost function to assess the cost of collision between two objects. As noted already, previous work with deformable objects requires the expensive calculation of the deformed geometry; our method, however, directly assesses the costs resulting from this deformation by observing the intersecting volume of objects in collision. Cost is computed for each voxel of the object (or objects) being moved in collision, and the sum of these per-voxel costs is the total cost of deformation for a given state.

Per-voxel cost is evaluated using Equation 3.1. Let C_i be the total deformation cost of voxel i, while $S_i(A)$ and $S_j(B)$ are the sensitivity parameters of voxel i in object A and j in B, respectively. Similarly, $D_i(A)$ and $D_j(B)$ are the deformability parameters of voxels i and j.

$$\begin{aligned} C_i(A,B) = & \frac{D_i(A)}{D_i(A)+D_j(B)} * S_i(A) \\ & + \frac{D_j(B)}{D_i(A)+D_j(B)} * S_j(B) \end{aligned} \tag{3.1}$$

Intuitively, this cost function assigns cost based on the weighted combination of costs incurred by both objects. If both objects have the same deformability, each will contribute equally to the total cost (if $S_i(A) = S_j(B)$), while in cases of varying deformability, the "softer" object with higher deformability contributes more to the total cost. In cases of hard-on-soft or soft-on-hard collision where one object is completely rigid, only the soft object being deformed contributes to the total cost.

A notable special case of Equation 3.1 exists if both $D_i(A)$ and $D_j(B)$ are zero (meaning that voxels i in A and j in B are both rigid), in which case $C_i(A,B)$ becomes undefined. This property is used for implicit collision detection in our planner, as our implementation of the cost function returns NaN

in these cases, which allows states resulting in rigid body collisions to be eliminated. Similarly, using NaN for sensitivity values can also be used to make certain states explicitly infeasible.[1]

3.2.3 Discrete Path Planning

For low-dimensional problems, we use a planning algorithm based on the well-known A* search algorithm [80]. A* is both complete, meaning that it will always find a solution if it exists, and optimal, meaning that it will always find the minimum-cost path to the solution.

In its conventional form, A* orders states based on the combined value of $g(s)$, the *cost* of moving to state s, and $h(s)$, the *heuristic* value of s. A* alone is sufficient for use in rigid environments in which states are either feasible and free of collision, or infeasible due to collision; however, A* is insufficient to handle planning in deformable environments with feasible collisions.

Algorithm 1 Cost function for our A* planning algorithm

```
procedure FVALUE(s, p)
    return (1 − p) × (h(s) + g(s)) + p × DEFORM(s)
procedure DEFORM(s)
    cost ← s.parent.cost
    for each point A in s.shape do
        B ← LOOKUP(A)
        cost ← cost + C(A, B)
    return cost
```

Thus, we adapt the A* algorithm to account for the *deformation cost* in addition to the path length cost, as seen in the DEFORM and FVALUE functions in Algorithm 1. The DEFORM function computes the total cost of deformation in a given state using $C(A, B)$, our cost function shown in Equation 3.1 for all voxels in the moving object. LOOKUP transforms a given point in the object being manipulated into the planning environment and returns the corresponding voxel. FVALUE, which implements the classical $f(s) = g(s) + h(s)$ in A*, is extended to incorporate the deformation cost of the path to s, which is returned by DEFORM. Here, $g(s)$ is the path length from the start to state s, and $h(s)$ is the euclidean distance from s to the goal.

It is important to point out that we have not simply appended deformation cost to the overall state cost. Instead, we have incorporated the concept of Pareto-optimality for paths discussed in [81], which allows us to compute paths with varying definitions of optimality. This control is introduced with the parameter p, which weighs the deformation against path length. Intuitively, low values of p induce greater deformation if doing so will result in a shorter path, as they increase the relative penalty of path length. High values of p may result

[1] As defined in the IEEE Floating Point standard, NaN "poisons" calculations, so a single voxel cost of NaN results in a total cost of NaN.

in lower deformation at the expense of longer paths. The two boundary cases of p, namely $p = 0$ and $p = 1$, result in conventional A* (ignoring deformation) and best-first search considering only deformation, respectively. Note that using $1 - p$ to scale the heuristic is necessary to ensure that the heuristic remains admissible.

In practice, selecting values for p close to 0 may result in undesirably high deformation and values close to 1 may produce unnecessarily long paths. In cases where no deformation-free path exists, the effect of p is dependent on the total deformation encountered and the cost parameters of the deformable objects. In practice, the paths for the range of p values can be presented to the user, allowing the user to select from the Pareto front.

3.2.4 Sampling-based Planning

For higher-dimensional problems, we evaluate both the T-RRT algorithm [78] and the GradienT-RRT algorithm [79] with our representation. Both are probabilistically-complete sampling-based planners suitable for cost-space planning in high-dimensional spaces. Our choice of these planners instead of the better-known RRT* [9] is because T-RRT has been shown to outperform RRT* [82, 83] in higher dimensions and because GradienT-RRT is an extension of T-RRT specifically intended for narrow low-cost regions such as those encountered in many scenarios where objects need to be deformed to complete a task.

The T-RRT algorithm, unlike the basic RRT, uses cost to control the addition of nodes to the tree. Addition of nodes is a function of the cost of the new node, the cost of its parent, and the distance between them (see Algorithm 2 in [78] for details). New nodes of lower cost than their parents are automatically added. Higher-cost nodes are added to the tree with probability dependent on the cost increase and the current *temperature*. Expansion behavior is primarily controlled with the $nFailMax$ parameter, which specifies the number of unsuccessful extensions required to increase temperature. Effectively, $nFailMax$ can be used to trade between cost and planning time – lower values will result in more rapid expansion (and thus faster planning), while higher values will result in lower cost solutions, usually at the expense of longer planning time.

The GradienT-RRT algorithm is designed to address particular shortcomings of T-RRT, namely in the inability of T-RRT to follow narrow valleys in the cost-space. Instead of simply rejecting higher-cost nodes, GradienT-RRT adjusts them using the gradient of the cost function. If these new nodes result in lower cost, they are then added to the tree. GradienT-RRT has been previously applied to a range of cost-space problems including those with workspace, task-space, and configuration-space costs. However, GradienT-RRT requires a function that computes the gradient ∇q of the cost function at configuration q in addition to the cost of that configuration. Our approach to computing the gradient derives from previous work planning for workspace uncertainty [79].

The gradient for a given state is computed as shown in Equation 3.2 in a similar manner to that used in [79].

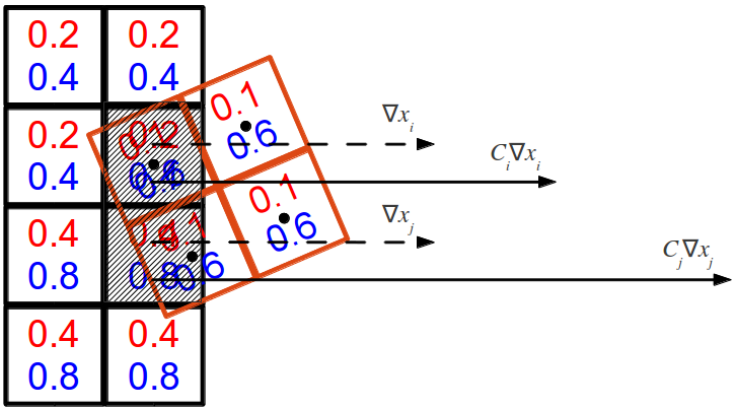


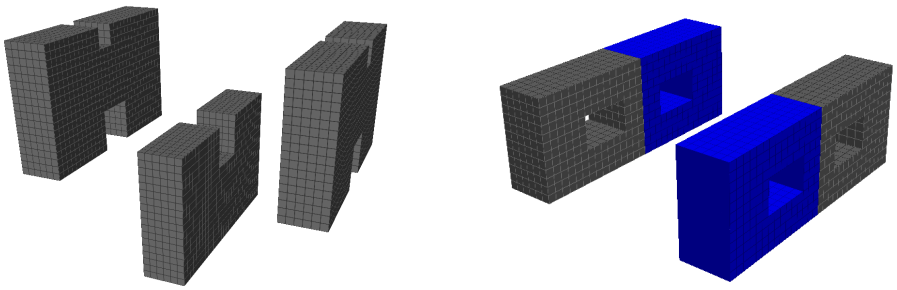
Figure 3.3: Workspace gradient computation: initial per-voxel gradients $\nabla x_i, \nabla x_j$ (dashed arrows) are computed using a signed distance field in GradienT-RRT. These gradients are scaled by the cost function evaluated at the voxel of intersection (shaded) to produce the final $C_i \nabla x_i, C_j \nabla x_j$ (solid arrows).

$$\nabla q = \mathbf{J}(q, x_1, x_2, ...)^T [C_1 \nabla x_1^T, C_2 \nabla x_2^T, ...]^T \tag{3.2}$$

Here, a workspace gradient ∇x_i is computed for each voxel x_i of the robot that intersects an obstacle at configuration q. As this workspace gradient only reflects penetration of filled voxels, we multiply the magnitude of the gradient for each voxel with the cost computed from our aforementioned cost function C_i as shown in Figure 3.3. We use the Jacobian $\mathbf{J}(q, x_1, x_2, ...)$, a composition of the Jacobians for each point in the intersection, to convert this workspace gradient to the C-space gradient ∇q needed for GradienT-RRT. For both T-RRT and GradienT-RRT planners, edge cost is simply the change in cost between a node and its parent. Note that we do not currently have an equivalent to p for our sampling-based planners, as the paths produced by T-RRT and GradienT-RRT are not guaranteed to be optimal (though they have low cost in practice).

3.3 Results

We have applied our methods to both simulation-only environments and simpler physical environments manipulated by a PR2 robot. For low-dimensional problems, we have implemented a planner integrated with ROS [84], which provides both visualization for planning and testing, control of the PR2, and the interface to our custom deformation tracking system. For higher-dimensional problems, we have modified the existing GradienT-RRT planner in the OpenRAVE [85] planning environment and implemented a validation environment using the Bullet physics simulation engine [86]. We show the performance of



(a) Triple-wall (b) Multi-cost

Figure 3.4: Simulation environments (a) three walls with a deformation-free path, (b) two walls with multiple sensitivity values – blue sections have half the sensitivity of gray sections.

these planners on several problems and report time and cost results. We also present a way to calibrate our model.

3.3.1 Low-dimensional Planning

We first demonstrate the capabilities of our representation in low-dimensional environments using our A*-derived planner. We use a set of simulated environments and objects with different combinations of hard and soft material properties. The environments are modelled at a resolution of 8mm, for a total size of 54000 voxels. Due to the well-known performance problems of applying A* directly to high-dimensional problems, we limit planner control to 3D translation of the object.

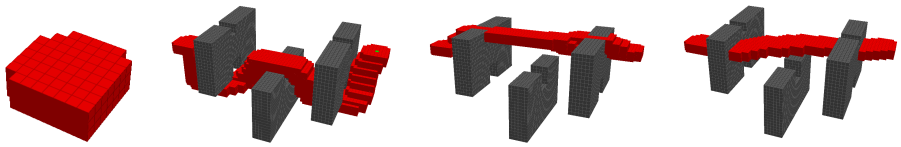
Environment

We have built two simulation environments, shown in Figure 3.4, that demonstrate the capabilities of our low-dimensional planning method. The first of these environments provides a set of distinct pareto-optimal paths with decreasing path lengths and increasing cost, while the second environment illustrates the control provided by the sensitivity parameter (e.g., $S_i(A)$) of our representation. With both of these environments, the behavior of the planner can be controlled using p. Additionally, due to the simplicity of our representation, it is trivial to change these environments to reflect all four combinations of hard/soft obstacles and robot, and we report results for all of these cases.

The first environment, ("triple-wall"), allows both deformation-free and deformed paths. By varying p, we are able to control the balance between deformation and path length. Additionally, due to the existence of a deformation-free solution, it can be simulated as a fully rigid environment instead, and we provide this for comparison.

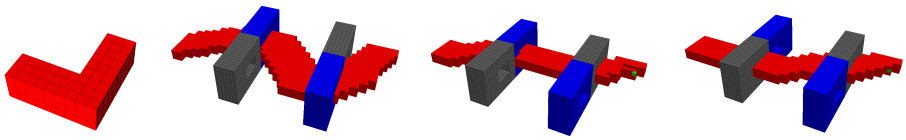
The second environment, ("multi-cost"), illustrates the capability of our deformability and sensitivity representation. Both black and blue obstacles in this environment exhibit the same nominal physical properties (captured by deformability). However, we assign lower *sensitivity* to the blue obstacles to indicate that deformation of them is less severe. As before, using p we can tune the behavior of the planner to produce paths that deform either or both blue and black obstacles. While simple, this environment illustrates the advantage of our representation, namely that it combines both physical and qualitative properties of objects that are difficult to capture using purely mechanical models.

Testing



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

Figure 3.5: Path classes for the Triple-wall simulation environment (soft) using the robot (hard) shown (a) with the swept volume of the robot shown in red (b) deformation free path: length = 94, $p = 0.7$, deformation = 0, (c) medium deformation path: length = 65, $p = 0.01$, deformation = 683, and (d) highest deformation path: length = 61, $p = 0.0$, deformation = 1062.



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

Figure 3.6: Path classes for the Multi-cost simulation environment (soft) using the robot (hard) shown (a) with the swept volume of the robot shown in red (b) lowest deformation path: length = 73, $p = 0.7$, deformation = 81, (c) medium deformation path: length = 58, $p = 0.01$, deformation = 159, and (d) highest deformation path: length = 57, $p = 0.0$, deformation = 310.

Using the triple-wall environment, we have run the planner for all combinations of hard and soft properties and values of p ranging between 0 and 1. Three distinct examples of these paths can be seen in Figure 3.5. Notably, only very low values of p, such as $p = 0.01$ result in *any* incurred deformation for this test environment. This is due to the imbalance between cost incurred due to path length and cost of deformation – the optimal non-deformation path through

the environment has a length of approximately 94, while a deformation-causing path of length 61 incurs a deformation cost of 1062. In contrast, the multi-cost environment lacks a deformation-free path. As before, we have tested the planner with a range of p values. Several examples of paths with corresponding values of p used to produce them can be seen in Figure 3.6.

Performance

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

Figure 3.7: (a) Vision system for deformation tracking, (b) the three Pareto-optimal path classes, (c),(d),(e) execution of the shortest path through the physical test environment, as seen by the forearm camera of the PR2.

For each combination of hard and soft environment and robot, we ran our discrete planner 101 times on each environment, increasing p from 0.0 to 1.0 in 0.01 increments. Planning times for the two simulation environments (triple-wall and multi-cost) average approximately 14.4 seconds and 2.3 seconds, respectively, for these experiments. A notable outlier in terms of run time exists for very low values of p in cases with soft environments and/or soft robots, in which a solution is found in under 2 and 1 seconds, respectively, because the cost of deformation is effectively ignored in these cases. In general, while computation of deformation adds comparatively little overhead to the state evaluation process, planner performance is worse with fully deformable environments or fully deformable robots with p values above zero, as there are no longer any *infeasible* states that can be eliminated as would be the case with rigid environments. As a result, there are more states to consider.

3.3.2 PR2 Testing

We have also built a test environment to evaluate our cost function and demonstrate the performance of our planner when applied to a physical environment. This test environment is similar in concept to the triple-wall simulation environment discussed already, containing a hard robot and soft obstacles, which has been simplified to allow assessment of deformation. As full tracking of deformation in the physical world is currently very difficult to accomplish, our test environment allows us to sense physical deformation.

PR2 Implementation

Our test environment consists of obstacles built out of deformable foam blocks on a rigid backing. These blocks are attached to the backing to prevent rotation

or movement without affecting the deformable behavior of the blocks' exterior (see Figure 3.7(a)). For planning purposes, the test environment is represented at 5mm resolution, as this offers a balance between accuracy and fast planning times. Planning times for the test environment average 1.2 seconds with a maximum of 1.7 seconds, requiring the evaluation of at most 4818 unique states.

To represent the simulated robot in our test environment, we use the red cylinder shown in Figure 3.7(b) which is moved through the environment by a PR2 robot. For our tests, paths planned in the simulator are converted into pose-space trajectories in the PR2's reference frame. Using the provided inverse-kinematics software, we convert these pose-space trajectories to joint-space trajectories which are then executed using the PR2's provided joint trajectory controllers, while the base of the robot remains at a fixed location.

Deformation Tracking

To assess the accuracy of our cost function, we use a vision-based tracking system shown in Figure 3.7(a) to provide ground-truth values for deformation. This vision system consists of a camera mounted above the test environment which measures the visible deformation of the foam environment. Our test environment is specifically designed so that only the areas made up of deformable foam are visible to the camera, as these are the only areas in which deformation may occur.

Testing

Using three Pareto-optimal paths produced by the planner shown in Figure 3.7(b), we executed each path 12 times through the environment with six executions in either direction; snapshots from the execution are shown in figures 3.7(c-e). Notably, the deformation-free path of $p = 0.5$ caused a non-zero measured deformation – this was due to A* planning paths that closely follow the shape of obstacles, which results in interaction between the surface of the foam environment and the plastic object. Overall, however, measured deformation and observed behavior of the foam strongly correlates to that expected from the planner.

Calibration

An additional role of the deformation tracking system is to calibrate the costs returned by the planner's cost function to the costs measured by the tracking system. To calibrate, we calculated the ratio between planned and measured deformation (in pixels) for each point of the three paths, and used the mean of this data to scale the cost computed by our cost function. For the two Pareto-optimal paths in Figure 3.7(b) with planned deformation, the planner computed cumulative path costs of 581 and 1240 in units of our cost function. We measured cumulative path deformation of 19700 and 56200 in units of pixels. Applying our calibration, the planner costs are 23000 and 51000 pixels, respectively, which

are within 17% and 9% of the measured values. Inaccuracy in the calibration occurs at points of starting and ending deformation; we believe this results from the discretization of the planner and the material properties of our foam test environment.

3.3.3 Higher-dimensional Planning

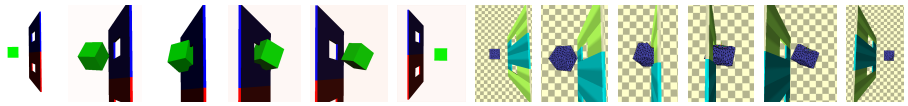


Figure 3.8: 6-dimensional planning: (a-f) Execution of a planned trajectory in the OpenRAVE environment used for planning, (g-l) execution of the same trajectory in the Bullet physics simulator.

To demonstrate the suitability of our representation for higher-dimensional problems that cannot be reasonably addressed with discrete planning approaches, we use the T-RRT and GradienT-RRT planners in the environment shown in Figure 3.8.

Environment

In this environment, the vertical wall is rigid and the cube robot is deformable. The vertical wall is largely symmetric; however, the hole in the blue half is 12.5% larger and thus presents a lower-cost path. The robot has 6 degrees of freedom (translation and rotation), but is neither capable of moving around the wall nor passing completely through the wall, as the very center of the robot is rigid. Rather, a valid solution must pass through one of the two holes, both of which are smaller than the robot. These holes pose two particular challenges to the planning algorithm as each presents both a "narrow passage" and a "cost-space chasm" [79] – i.e., a narrow area of low cost.

Testing

Using the aforementioned testing environment, we can produce paths such as those shown in Figure 3.8. As already discussed, the GradienT-RRT planning algorithm is especially designed to navigate the low-cost C-space region resulting from the hole in the wall, however, neither it nor T-RRT is particularly designed to address the problem of *entering* this region.

In our initial tests, both T-RRT and GradienT-RRT failed to compute any paths given a reasonable time limit. As a workaround, we added virtual padding to the obstacle when creating the discretized cost-space. This effectively increases the volume of the obstacle and produces a smoother gradient instead of a sharp boundary. Counter-intuitively, while this padding makes the "narrow

passage" of entry narrower, it makes the *volume* of the cost-space chasm larger, which increases the probability of sampling within it. Notably, it also increases the area of "useful gradients" that push the algorithm towards the hole. For paths such as that shown in Figure 3.8, padding increased the length of the cost-space chasm from 0.1m to 0.7m for both holes.

Performance

	T-RRT		GradienT-RRT	
nFailMax	10	20	10	20
Planning time (s)	112(45.6)	736(260)	20.6(10.7)	175(69.4)
Planned cost	515(82.3)	478(56.9)	714(326)	563(164)
Calib. cost (m^3)	3.04(0.48)	2.82(0.33)	4.21(1.90)	3.32(0.96)
Sim. cost (m^3)	2.97(1.46)	2.63(1.28)	4.10(2.04)	3.48(1.29)

Table 3.1: Performance data [mean(std. dev.)] for T-RRT and GradienT-RRT planners. Cost given is the integral of costs incurred at each state in a trajectory.

We ran both planners 30 times in our test environment, each with $nFailMax = 10$ and $nFailMax = 20$. The results of these trials are shown in Table 3.1. GradienT-RRT produced solutions in all 30 trials for both values of $nFailMax$, while T-RRT failed to find a solution in 1200 seconds for 8 of 30 trials with $nFailMax = 20$. GradienT-RRT is significantly faster than T-RRT with the same parameters, however, T-RRT produces lower-cost paths and always traverses the lower-cost hole when it returns a solution. At $nFailMax = 10$ and $nFailMax = 20$, GradienT-RRT planned paths through the higher-cost hole seven and one times, respectively. The high standard deviation for the cost produced by GradienT-RRT is the result of these paths through the higher-cost hole.

As is clearly visible in Table 3.1, T-RRT and GradienT-RRT work best with different values of $nFailMax$. GradienT-RRT, by virtue of "greedily" following the cost-space gradient, requires a higher value (e.g., 20) to discourage planning through the higher-cost hole. T-RRT, on the other hand, requires longer planning times for a given $nFailMax$ but produces lower-cost paths than GradienT-RRT with the same parameters. Notably, both planners produce "corner-first" trajectories for the cube, demonstrating that our planners take advantage of the rotational degrees of freedom to reduce deformation.

3.3.4 Simulator Validation

Given the difficulty of assessing real-world deformations, our simulation environment provides an alternative to real-world testing – albeit one limited by the accuracy of the simulator. To assess the paths produced with the T-RRT and GradienT-RRT planning algorithms, we have implemented a validation environment using the Bullet physics simulation engine to match the OpenRAVE planning environment shown in Figure 3.8. This validation environment provides

soft-body physics to simulate the physical interactions between the deformable cube and the wall. The deformable robot is modelled using a tetrahedral mesh anchored to a small rigid body. This setup allows the soft robot to be "towed" along the path produced by the planner. Deformation is assessed by computing the current volume of each tetrahedron in the tetrahedral mesh (in m^3) and comparing the total volume against that of an undeformed reference mesh.

Our calibration strategy is similar to that used previously in Section 3.3.2. We calibrated the raw costs returned by the planner for each of the 112 trajectories to produce the calibrated cost values shown in Table 3.1. Discrepancies between the calibrated planner cost and the simulator cost are largely due to "mangling" of the deformable cube – i.e., it does not return to its original shape after deformation, which is an artifact of the Bullet simulator. This effect can be seen in Figure 3.8(l).

3.3.5 Representation Performance

Cost assessment using our cost function offers significant performance improvements over cost assessment using physical simulation. A single cost assessment for the deformable cube discussed in Section 3.3.3, modelled with 1000 voxels in our representation, takes an average of 84 microseconds, regardless of the state of the deformable object. In comparison, the same cost assessment done using the Bullet physics simulator (see Section 3.3.4), with the cube modelled with 400 tetrahedrals, takes an average of 4 milliseconds when the cube is in contact with a rigid obstacle, and 16 milliseconds when the cube is in contact with a deformable obstacle. Not only is using our representation significantly faster (almost 50 times so in hard-on-soft and 200 times so in soft-on-soft), but Bullet at these settings – tuned for a balance between simulation quality and speed – exhibits severe mangling and distortion of the deformable object during and after deformation. Tuning parameters in favor of higher simulation quality results in an even greater performance gap, while tuning them in favor of faster simulation results in prohibitively poor simulation quality.

3.4 Conclusions

We have proposed a new method of representing deformable objects that allows both physical and qualitative properties to be captured in a voxel-based representation. Using this representation, we have designed a cost function that directly assesses the severity of deformation without expensive physical simulation or computation of deformed geometry. This cost function is particularly suitable for motion planning, and we have demonstrated its application to both discrete motion planning in low dimensions and sampling-based motion planning in higher dimensions. We show that our methods can generate paths that minimize deformation in both simulated and physical environments with either hard and soft robots in either hard and soft environments. In addition, using both a physical test environment and a soft-body simulation environment, we

have demonstrated methods for calibrating our object representation to match observed object behavior.

Chapter 4

Reproducing Expert-Like Motion in Deformable Environments Using Active Learning and IOC

4.1 Introduction

Modeling deformable objects is a difficult problem; models must not only capture the geometry (undeformed and deformed) of objects (itself a very difficult problem), but should also capture the sensitivity of the object. This qualitative aspect is critical for deformable environments, as it allows a motion planner to distinguish between multiple objects with similar physical properties but with different qualitative characteristics. An important example of this occurs in surgical robotics; while multiple organs and tissues may have similar physical properties, some parts of the body are significantly more sensitive than others. Without accounting for sensitivity, motion planners can produce paths that could cause unnecessary injury.

The motion planning methods introduced in the previous chapter use a voxel-based representation of deformable objects in which each voxel has two parameters. The first parameter, *deformability*, captures physical properties of the rigidity of the material. The second parameter, *sensitivity*, captures the qualitative significance of deforming the object. Together, these parameters are used in a cost function that provides a cost of deformation that can be used in cost-aware motion planners.

While the deformability parameters are directly related to material properties, setting the sensitivity parameters is more difficult, as they capture a range of object characteristics. Setting them by hand is time-consuming and error-prone, as incorrect sensitivity values can produce unwanted planner behav-

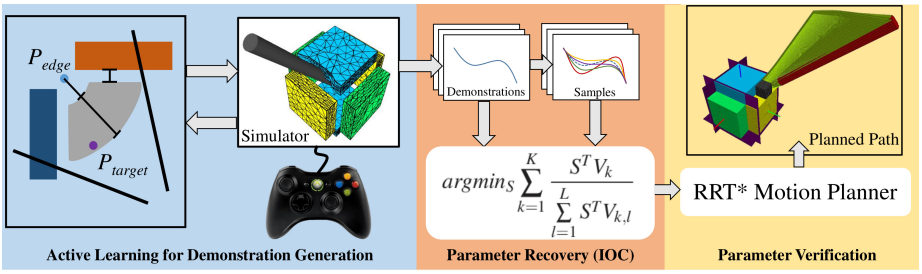


Figure 4.1: Diagram of the three stages and main components of our framework.

ior. More problematically, setting these parameters for practical environments requires both domain knowledge and the ability to mathematically represent that knowledge such that the planner will perform well. Instead, we propose a framework for automatically learning and validating these parameters from expert demonstrations. For example, a surgeon can demonstrate the optimal path for inserting a probe, and we can use this demonstration to find the sensitivity values of organs around the path. For completeness, in Appendix A we show a simple method for learning deformability parameters in a given simulated environment.

Our framework consists of three parts: (1) Automatic generation of demonstration tasks that prompt the user to provide informative demonstrations using a novel active learning process; (2) Recovery of object sensitivity values using *Path Integral Inverse Reinforcement Learning* (PIIRL) Inverse Optimal Control (IOC) techniques [8]; and (3) Reproduction of the demonstrated behavior using the RRT* asymptotically-optimal motion planner [9] with a key modification that allows us to check for punctures of deformable objects.

This approach offers two main advantages over existing similar techniques. First, by using sampling-based techniques for IOC that avoid the need to solve the forward problem as well as sampling-based asymptotically-optimal planners, our framework is applicable to higher-dimensional problems than approaches such as LEARCH [6], which are limited by the need to repeatedly compute optimal paths to recover the cost function. Second, our proposed method for automatically generating demonstration tasks for experts to perform reduces the number of demonstration tasks needed to capture the desired behavior and removes the need for domain knowledge to generate these tasks by hand. Finally, to our knowledge, IOC has never before been applied to the problem of learning deformable object parameters.

In our experiments in simulated and physical test environments we show that, despite the limitations inherent in asymptotically-optimal sampling-based planning, the recovered sensitivity parameters allow motion planners to reliably reproduce behavior demonstrated by expert users. We also present experiments which show the generalization capabilities of our method.

4.2 Problem Statement

Let τ represent the path of a rigid object (i.e., the robot) through an environment composed of n deformable objects $E = O_1, O_2, ..., O_n$. Representing τ with a discrete sequence of configurations, we assume the cost of executing τ is a function of the form $C(\tau) = \sum_{k=1}^{|\tau|} \sum_{i=1}^{n} D_i S_i V_i(\tau_k)$, where $V_i(\tau_k)$ is the volume of deformation of O_i that results from placing the rigid object at the kth configuration of path τ, D_i is the deformability of O_i, and S_i is the sensitivity of O_i. We focus on learning the S_i parameters, so we assume $D_i = 1 \, \forall i$, though our methods work with any known D. Note that while sensitivity parameters S can be set per-voxel in our representation, we simplify the problem of recovering sensitivities by assuming that each object has uniform sensitivity.

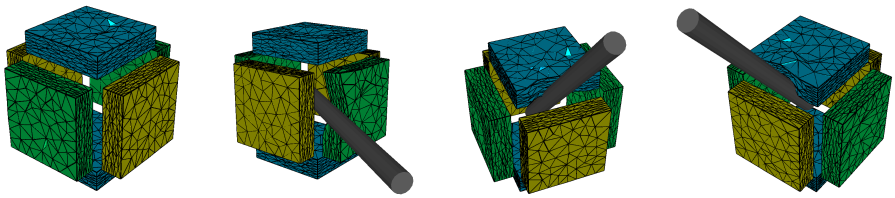
S represents the ground-truth sensitivities of the objects. We seek to generate a set of learned parameters $\hat{S}$ from a set of demonstrations, such that these $\hat{S}$ can be used in a motion planner to produce similar behavior to the demonstrations. Obtaining the true S from demonstration is not possible in general, as a demonstration can, at best, encode only the ratios between different elements of S and not their magnitudes. Thus it is not meaningful to compare S to $\hat{S}$ directly. A more informative comparison is how well a planner imitates demonstrated behavior when planning with $\hat{S}$. Thus we evaluate our method in terms of the cost of the path produced by our framework. Therefore the quality of $\hat{S}$ relative to the ground truth is evaluated as $E(\hat{S}, S) = |C_S(\tau_d) - C_S(\tau_{planned}(\hat{S}))|$, where τ_d is a path demonstrated for a given task, $\tau_{planned}(\hat{S})$ is a path planned for the same task using the sensitivities $\hat{S}$, and the cost function $C_S(\cdot)$ is evaluated using the ground-truth sensitivities S.

4.3 Methods

We have developed a framework for recovering sensitivity parameters for deformable objects, as illustrated in Figure 4.1. Below we describe each of the four components in detail.

4.3.1 Capturing Demonstrations

Like all IOC problems, our approach requires demonstrations. In our case, demonstrations are captured in a simulation environment using a physics simulator to simulate deformable objects. Our demonstration task consists of inserting a cylindrical probe between deformable objects to reach target points distributed across the environment, as illustrated in Figure 4.2. The user attempts to minimize contact with more sensitive objects (shown in yellow and green) compared to less sensitive objects (shown in blue). We record the demonstration trajectory along with the features of that trajectory, which are the total amounts of deformation of each object. While outwardly simple, the problem of probe and needle insertion between deformable objects such as this is common in medical tasks [12] and a subject of previous research in robot motion [12, 1],



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

Figure 4.2: Example demonstration tasks for our 6-object test environment, shown with the probe reaching the target. (a) Low sensitivity objects L_1, L_2 (blue), medium sensitivity objects M_1, M_2 (green), and high sensitivity objects H_1, H_2 (yellow). (b,c,d) Goal configurations for three automatically generated tasks.

however, none of this work has explored learning qualitative properties of deformable objects to determine higher-level behavior. In addition to capturing demonstrations, we use this simulation environment to compute feature vectors for demonstration and sample paths.

Each demonstration we capture can be parametrized as a *demonstration task* by a starting pose of the probe P_{start}, a target point P_{target} the user must touch with the probe tip, and a set of "collision planes" $Cplanes$, hyperplanes that constrain the motion of the probe. As shown in Figure 4.3, the hyperplanes approximate a funnel that guides the user towards the target point and restricts which objects the user can contact with the probe. These hyperplanes are added to constrain the user to producing demonstrations that capture the relative difference in sensitivity between the accessible objects. In our experience, without the hyperplanes users sometimes produce demonstrations that deform only the globally least-sensitive object(s) instead of capturing sensitivity relationships between neighboring objects.

While we attempt to capture optimal demonstrations, in practice users may provide slightly sub-optimal demonstrations. We attempt to correct for this using a local optimizer that optimizes each demonstration. This method generates a set of random sample trajectories around the demonstration trajectory and replaces the demonstration trajectory with any of the random samples with strictly dominating deformation (i.e., the random sample deforms all objects less than or equal to the demonstration).

4.3.2 Active Learning

We can capture demonstrations and compute features for demonstrations and samples needed for PIIRL, however, this leaves two problems to address: how to generate demonstration tasks for the user to complete, and how many demonstrations must be collected. Clearly, the accuracy of recovered sensitivity values depends on the quality of the demonstrations provided. For example, if an object has zero feature values in both demonstrations and the trajectory samples around the demonstrations used by PIIRL, we cannot recover a meaningful

Algorithm 2 Demonstration task collection algorithm

procedure CollectDemonstrations(A)
  $G \leftarrow \{\emptyset, \emptyset\}$
  $O_1 \leftarrow argmax_{o \in E}$ degree($A(o)$)
  $O_2 \leftarrow argmax_{o \in \text{neighbors}(A(O_1))}$degree ($A(o)$)
  $G \leftarrow G \cup$ CollectSingleDemonstration(O_1, O_2)
  while $\{o \in E | o \notin G_v, \text{degree}(A(o)) > 0\} \neq \emptyset$ **do**
    $O_1 \leftarrow argmax_{\{o \in G_v | \text{neighbors}(A(o)) \setminus G_v \neq \emptyset\}}$ depth($G(o)$)
    $O_2 \leftarrow argmax_{\{o \in \text{neighbors}(A(O_1)) \setminus G_v\}}$ degree($A(o)$)
    $G \leftarrow G \cup$ CollectSingleDemonstration(O_1, O_2)
    $G \leftarrow$ EnsureRanking(G)
  return G
procedure EnsureRanking(G)
  for $O_1 \in G_v$ **do**
    for $\{O_2 \in G_v |$ depth($G(O_2)$) $\geq$ depth($G(O_1)$)$\}$ **do**
      if NoDirectedPathExists(O_1, O_2) **then**
        if DirectlyComparable(O_1, O_2) **then**
          $G \leftarrow G \cup$ CollectSingleDemonstration(O_1, O_2)
          return EnsureRanking(G)
  return G
procedure CollectSingleDemonstration(O_1, O_2)
  $P_{target}, P_{edge}, C_{planes} \leftarrow$ GenerateTask($O_1, O_2, T_{clearance}, T_{range}$)
  $D_v, D_e \leftarrow$ GetDemonstrationFromUser($P_{target}, P_{edge}, C_{planes}$)
  return (D_v, D_e)

sensitivity value for the object; e.g., if all demonstrations entered through the forward half of our cube environment, no features would be available for objects on the reverse. A different, but equally problematic, issue occurs when features have been collected for every object, but the demonstrations are "unconnected"; for example, in an environment $E = \{O_1, O_2, O_3, O_4\}$, if features have been collected for demonstrations between O_1, O_2 and O_3, O_4, but not for O_2, O_3, the optimizer cannot determine if O_1 and O_2 are more or less sensitive that O_3 and O_4. Thus, we need to ensure that sufficient demonstrations have been collected.

The conservative solution is to require a demonstration for every pair of adjacent objects, however, this can result in a large number of demonstrations. For our test environment shown in Figure 4.2, 12 demonstrations would be required to capture the relationship between every adjacent pair. We seek to reduce the number of demonstrations required.

Simply collecting demonstrations such that we observe a non-zero feature for each object is insufficient for accurate parameter recovery, rather, we must ensure that the demonstrations collected form a *ranking* of the objects in terms of sensitivity; i.e., that for objects $O_1, O_2 \in E$, $rank(O_1)$ is either less than, equal to, or greater than $rank(O_2)$ if the objects are comparable. Rankings are derived from demonstrations collected between adjacent objects; the preferen-

tially deformed object receives a lower ranking than the preferentially avoided object. Rankings are not comparable in certain cases, such as between objects on the opposing faces of our test environment, in which it impossible to perform a demonstration between the two objects, and they cannot be ranked via a combination of other demonstrations.

Demonstrations are collected using Algorithm 2, which takes A, the set of object adjacencies in E, and iteratively collects demonstrations until there are no more useful demonstrations to perform. This algorithm captures preference relationships between objects by building a directed graph G. The nodes in G represent objects in the environment and the directed edges point from the less-sensitive object to the more-sensitive one. Initially, G contains no nodes or edges, and each demonstration adds an edge and 0, 1, or 2 nodes. The key to the algorithm is determining which demonstration (and thus which edge) should be queried next.

The algorithm uses the structure of G at the current time as well as a heuristic to decide which demonstration to query next. If the ranking between all objects in G is known, then the algorithm selects a new object to add to G (via a demonstration involving that object and one already in G). After adding a new object, the algorithm queries demonstrations until the ranking of all objects in the graph is again established (this is done in the EnsureRanking function). It then selects a new object to add, and so on, until no more objects can be added.

At each step where objects or edges are selected, we choose the object or edge based on connectivity heuristics. For new objects (i.e., those not already in G), we prefer those that are adjacent to as many other objects as possible. When picking objects already in G for a new edge, we prefer objects that have a higher "depth". Here depth(n) is the length of the longest directed path in G which ends at n. These heuristics bias the algorithm to create long chains of edges where possible, which is clearly beneficial for forming a ranked list; e.g., $rank(O_1) < rank(O_2) < rank(O_3) < rank(O_4)$ is a chain of three edges which gives a complete ranking of four objects.

Algorithm 2 is not guaranteed to produce the minimal set of demonstrations because it cannot foresee the results of future demonstrations. It frequently collects demonstrations early on that prove to be unnecessary in the final set of demonstrations. In pathological environments, Algorithm 2 may be forced to collect all possible demonstrations. However, in practice, we show that it reduces the number of demonstrations without significant impact on the recovered sensitivity parameters.

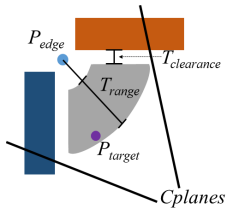


Figure 4.3: Our automatic demonstration task generator.

For each demonstration requested by Algorithm 2, we generate a new task using Algorithm 3. This algorithm is given a pair of target objects O_1, O_2, a target clearance $T_{clearance}$, and a target depth range T_{range}. First, the algorithm selects an "edge point", P_{edge} by randomly selecting a point on the medial axis between the two target objects. Using the edge point, the algorithm

Algorithm 3 Demonstration task generation algorithm

procedure GENERATETASK($O_1, O_2, T_{clearance}, T_{range}$)
  $P_{edge} \leftarrow$ GETEDGEPOINTBETWEENOBJECTS(O_1, O_2)
  $P_{target} \leftarrow \emptyset$
  while $P_{target} = \emptyset$ **do**
    $P_{sampled} \leftarrow$ SAMPLEINRANGE(P_{edge}, T_{range})
    if $clearance(P_{sampled}) < T_{clearance}$ **then**
      $P_{target} \leftarrow P_{sampled}$
  $Cplanes \leftarrow$ GENERATECOLLISIONPLANES(P_{edge}, P_{target})
  return $P_{target}, P_{edge}, Cplanes$

randomly samples nearby points T_{range} away from the edge point to select one that is "inside" [1] the environment and also at least $T_{clearance}$ away from an object, which it returns as P_{target}, the target point. Finally, a set of "collision planes" are generated to restrict the user's demonstration to the desired area. The parameter T_{range} ensures that the user must insert the probe sufficiently to cause deformations. Similarly, the parameter $T_{clearance}$ controls how close to an object the target point can be, and can be used to ensure that the target point itself is not in contact with an object (see Figure 4.3).

4.3.3 Parameter Recovery

Our approach to motion planning for deformable objects uses a "cost of deformation" to enable any motion planner that accounts for cost to produce plans that minimize deformation. We can frame the problem of imitating demonstration behavior as the problem of inferring the sensitivity parameters used to produce the demonstration. Assuming that the demonstration is optimal, this is the well-established problem of Inverse Optimal Control (IOC).

Using the PIIRL formulation of IOC, the cost function consists of a series of features $V = V_1, V_2, ..., V_n$ (in our case these are the amounts of deformation of each of the n objects) with corresponding sensitivities $S = S_1, S_2, ..., S_n$, such that the total cost of a configuration $C = \sum_{i=1}^{n} V_i S_i$, where the V_i can be computed using our physics simulator, but the optimal set of sensitivities S^* is unknown.

To find the best estimate of the optimal set of sensitivities $\hat{S}$, PIIRL requires a set of sample paths around each demonstration. Because the demonstrations are assumed to be locally optimal, all samples around a demonstration will be sub-optimal w.r.t. the unknown cost function. For K demonstrations and L

[1] To determine which points are "inside" the environment, we compute a "local maxima map" using the Signed Distance Field (SDF) of the environment. For each point in the SDF, we follow the gradient away from obstacles and record the location the gradient becomes zero (i.e., the local distance maxima). Points "inside" the environment have corresponding local maxima inside the bounds of the SDF, while points "outside" have local maxima corresponding to the bounds of the SDF. Intuitively, "inside" points have finite-distance local maxima reachable via the gradient, while for "outside" points, the local maxima are undefined.

samples for each demonstration, the optimal weights are obtained using the following minimization problem (a similar form of the minimization problem used in [8]), where V_k are the feature values for demonstration k, and $V_{k,l}$ are the feature values for sample l of demonstration k:

$$\hat{S} = argmin_S \sum_{k=1}^{K} \frac{S^T V_k}{\sum_{l=1}^{L} S^T V_{k,l}} \tag{4.1}$$

This minimization finds the sensitivity values $\hat{S}$ that maximize the margin between the cost of the demonstrations and the costs of their samples. Note that in our problem, $S > 0, V_{k,l} \geq 0$ and sample feature values $V_{k,l} \geq 0$, as all sensitivity values must be greater than zero and $V_{k,l} = 0$ implies $V_k = 0$ (since samples must be sub-optimal relative to their demonstrations). $V_k = 0$ implies that the demonstration k captures no information about any object and thus can be removed from the optimization so this condition will not occur.

In our experience, this modified form of the minimization recovers parameters with more distinct separation between low-, mid-, and high-sensitivity objects than the original form used in [8] using common function minimization tools such as those available in Matlab. Unlike previous work such as LEARCH [6], PIIRL does not rely on the specific configurations the demonstration path traverses; rather, only the corresponding feature values must be locally optimal in our cost function [87]. This makes it tractable to learn cost functions in high-dimensional spaces.

4.3.4 Recovered Parameter Verification

Once sensitivity values $\hat{S}$ have been recovered for each object in our test environments, we must verify that the recovered values allow our motion planner to imitate the behavior of the expert demonstrations. We attempt to perform each demonstration task using an optimal motion planner and comparing the planned path $\tau_{planned}(\hat{S})$ with the demonstration τ_d in terms of the true cost function $C_S(\cdot)$ using the ground truth sensitivity values S. In the previous chapter, we used the T-RRT and GradienT-RRT planners to efficiently produce paths in high-dimensional spaces; however, since these planners have no optimality guarantees, they are unsuitable for parameter verification. Instead, we use the asymptotically-optimal RRT* planner [9] with our deformation cost function. While we could use deformations measured via a physics simulator to compute cost during planning, our voxel-based deformation cost function is significantly faster, more stable, and detects object punctures and separation. To accurately mimic the demonstration tasks, the RRT* planner is provided with the same task-space target point to reach with the probe tip, rather than a goal configuration of the probe. Feasible configurations touching this target point can be sampled, and RRT* attempts to connect the tree to these goal states. As RRT* runs, it improves the path by reducing the deformation cost of

the path and by sampling and connecting to new, lower-cost goal states. Note that while RRT* is *asymptotically* optimal, for finite time it will not return *the optimal* path, so we expect paths reproduced with RRT* may be slightly higher-cost than their corresponding expert demonstrations, but should exhibit the same preferential deformation demonstrated by the expert.

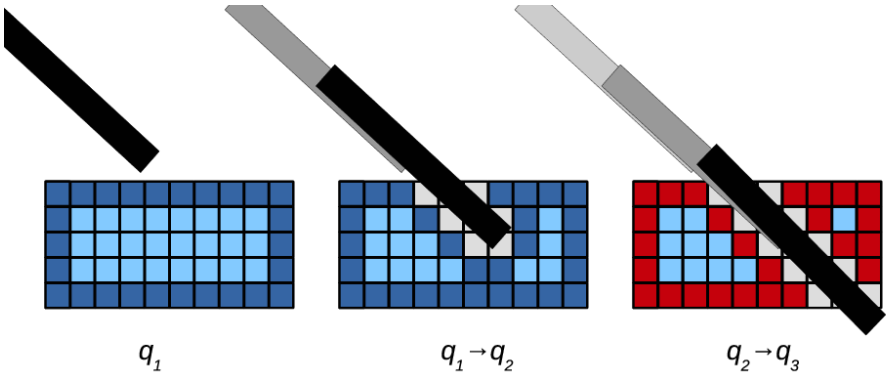


Figure 4.4: Illustration of puncture checking for an extension from configuration q_1 to q_3. As the surfaces are no longer connected (red), puncture has occurred and the $q_2 \rightarrow q_3$ motion is invalid.

In addition to integrating our existing cost function with RRT*, we have significantly improved the quality of planned paths by adding puncture detection to prevent paths from puncturing or cutting deformable objects. Puncture and cut detection is essential to planner performance; without it, planners can produce low-cost paths that pass directly though deformable objects. To prevent punctures and cuts, we check every extension of the tree in RRT* for puncture using an incremental variant of the algorithm introduced by Chen et. al. [88] for computing topological invariants on voxel grids. The original algorithm extracts the surface vertices from the voxel grid, and computes the connectivity of each surface vertex. Each surface vertex can be connected to between one and six neighboring surface vertices; let $M1$ be the total number of surface vertices with one connected neighbor, $M2$ the total with two neighbors, and so on. From these totals, Chen et. al. prove that the number of holes in the voxel grid is $n_{holes} = 1 + ((M5 + (2 * M6) - M3)/8)$.

Thus, checking for punctures can be implemented by removing the swept volume of the path of the probe from the voxel-based model of deformable objects used for motion planning, and then computing the number of holes to ensure that no new holes have been created by the path. Additionally, to prevent objects from being completely cut apart by the path, the overall connectivity of the surface voxels corresponding to each object are computed; if the surface vertices for an object form multiple disconnected groups, then the object has been cut apart by the path.

To efficiently perform these checks during the planning process, we incre-

mentally check for punctures with each extension and rewiring step of RRT* (see Figure 4.4). For testing a new edge from configuration q_1 to configuration q_2 the process is as follows: (1) retrieve the stored object surfaces corresponding to q_1, (2) update the object surfaces with the swept volume from q_1 to q_2, (3) compute the number of holes in each object surface (check for puncture), (4) compute the connectivity of each object surface (check for cuts), and (5) if no holes or cuts are encountered, store the updated surfaces corresponding to q_2. For every such check, we are effectively checking the entire path from the start configuration q_{start} to q_2 for punctures and cuts.

4.4 Results

We present results of testing our framework in a 3D simulated environment (5DoF probe insertion task) and in a physical planar environment (3DoF rigid object navigation task) using an industrial robot. We use the Bullet physics simulator [86] to provide an environment for capturing demonstrations and computing features, and the Open Motion Planning Library (OMPL) [89] to provide the RRT* planner used to verify the recovered sensitivity values. We show that our methods accurately recover sensitivity values that allow planners to imitate expert demonstrations. We also report on how the algorithm generalizes to a new task, where an obstacle is introduced into the environment, and report on the use of active learning for reducing the number of demonstrations required. Ideally, we would compare the performance of our framework with existing approaches such as LEARCH [6], however, these approaches require computing the true optimal path to perform IOC, which is intractable in the 5DoF probe insertion task.

4.4.1 Recovered Behavior

We first demonstrate the performance of our framework in the 3D simulated environment without using the automatic demonstration task generator, and show that our demonstration capture environment and parameter recovery process produce acceptable object sensitivity values. Using our RRT* planner, we show that the recovered sensitivities produce paths that imitate the expert demonstrations.

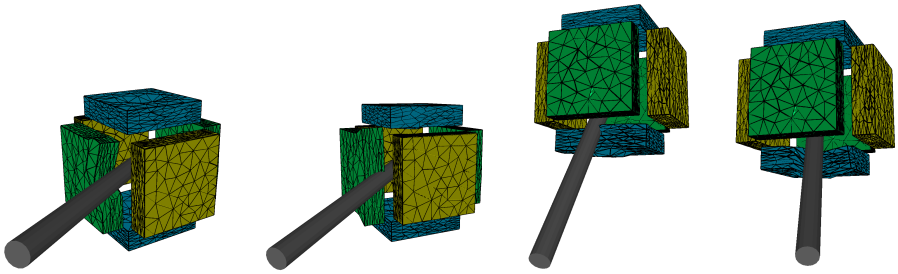
The test environment, as shown in Figure 4.2, consists of six deformable objects forming the faces of a hollow cube. These objects form three classes; each pair of opposing faces has the same sensitivity assigned, with the lowest sensitivity (L_1, L_2) shown in blue, an intermediate sensitivity (M_1, M_2) shown in green, and high sensitivity (H_1, H_2) shown in yellow. For testing purposes, the "true" sensitivity values of these objects are set as $L_1, L_2 = 0.2$, $M_1, M_2 = 0.4$, $H_1, H_2 = 0.8$. We use the true values to evaluate the quality of paths planned with the recovered sensitivity values, but they are unknown to our IOC method.

Using the conservative approach discussed in Section 4.3.2, 12 demonstrations were performed, one for each pair of adjacent objects. Several exam-

ples of these demonstrations can be seen in Figure 4.2 and Figure 4.5. While time-consuming, this approach ensures that sufficient demonstrations have been collected to capture the desired behavior. In these demonstrations, lower-sensitivity objects were preferentially deformed instead of higher-sensitivity objects.

Using the set of 12 demonstrations, we recovered the object sensitivity parameters using our parameter recovery process. We generated a set of 100 sample paths around each demonstration using a multivariate gaussian distribution using the process described in [87], which produces smooth noisy path samples around an initial path. Features for all demonstrations and samples were computed by executing paths in the demonstration capture environment, and all feature values were normalized relative to the highest feature value. Sensitivity parameters were recovered using the optimization problem in Equation (4.1); we used the function minimization tools in MATLAB to perform this optimization. For optimization, the lower bound of possible weight values was 0.1, and the upper bound was 1000, with the weights initialized to 500. The recovered sensitivity values were $L_1 = 0.10004$, $L_2 = 0.10092$, $M_1 = 2.8523$, $M_2 = 8.5683$, $H_1 = 958.92$, $H_2 = 999.51$. Note that both high sensitivity objects (H_1 and H_2) were avoided in all demonstrations, and thus received maximum weights in the optimization. Again, recovery of the true sensitivities is impossible and we must evaluate our method in terms of the cost of the path planned using the recovered sensitivities.

Recovered Parameter Verification



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

Figure 4.5: Examples of goal configurations from demonstrations (a,c) and corresponding goals of paths planned using recovered sensitivity values (b,d). Full paths are not shown for clarity.

Using the recovered object sensitivity parameters, we planned for all 12 demonstration tasks using RRT*. Table 4.1 compares the demonstrations with results for planning times of 30 and 60 minutes, with 30 and 15 trials of each, respectively. Figure 4.5 shows examples of demonstrated paths compared with paths produced by RRT*. As shown in the table, paths produced using the

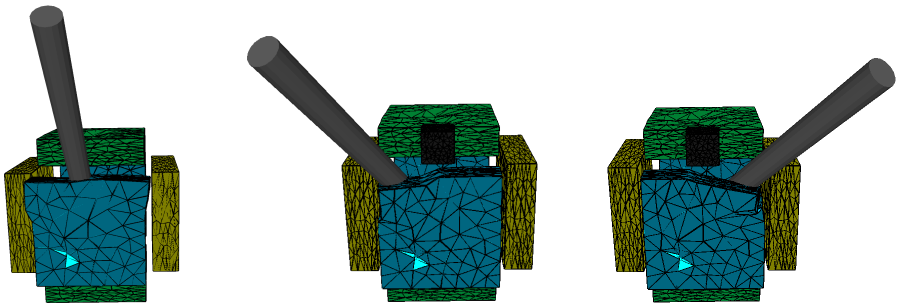
recovered parameters imitate the behavior of the demonstrations by deforming the same objects with similar amounts of deformation except for two demonstrations (namely 9 and 12) for which the planner found a path superior to the original demonstration. Note that due to the difficulty of the planning problem and the finite planning time for RRT*, we do not expect planned paths to exactly match the demonstrations. Two notable types of error resulted in sub-optimal plans, namely cases where planned paths clip the edge of higher-sensitivity objects, and cases where planned paths simply result in higher cost than the demonstration. In both cases, errors are indicated by high standard deviations; this is expected if a small number of the planned paths exhibit particularly sub-optimal behavior. These errors are caused by the limited time available to RRT*, which restricts the number of goal states sampled and the refinement of the path. Results for 60-minute planning times shown in Table 4.1 show that in most cases, increased planning time reduces these errors. Note that the high planning times used here are partially a consequence of our puncture test, which adds considerable computation in addition to the deformation cost function.

Generalization of Recovered Parameters

The importance of recovering sensitivity parameters is not to reproduce the demonstrations, since these could simply be replayed; rather, recovering the sensitivity parameters allows us to generalize the behavior displayed in the demonstrations to other tasks in the test environment. To demonstrate that the recovered sensitivity parameters generalize, we performed a set of tests shown in Figure 4.6. Starting from one of the demonstrations (demonstration task 6), we adjusted the target point and inserted rigid obstacles that block the demonstrated path. As shown in Figure 4.6, our planner produces paths that exhibit the same behavior as the demonstration path; while the new path differs from the demonstration and thus results in different cost, the preferential deformation of the blue object over the green one indicates that the expert's preference was correctly captured.

4.4.2 Automatic Generation of Demonstration Tasks

Using the same test environment, we tested our active learning method for automatically collecting demonstration tasks. Examples of these demonstration tasks are shown in Figure 4.2. Unlike the conservative approach discussed previously, which used demonstrations between all pairs of adjacent objects, the active learning method generates only enough tasks to form a ranking of all objects in the environment. We tested the active learning method in the same test environment as above and allowed it to select a subset of tests from the set of comprehensive demonstrations. Using this method, between 8 and 10 demonstrations were required to capture features for all objects, compared to the 12 used by the conservative approach. As before, 100 sample paths were generated around each demonstration, and sensitivities were recovered using

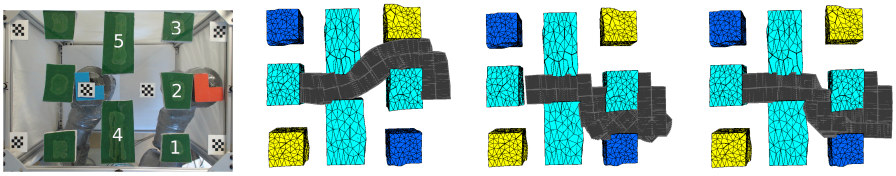


(a) L=7.27,M=0.0,H=0.0 (b) L=8.20, M=0.0, H=0.0 (c) L=21.24, M=0.0, H=0.0

Figure 4.6: Paths planned to show the generality of recovered sensitivity values, (a) goal configuration of demonstration 6, and (b)(c) two goals of paths planned with target points offset from the center of the environment when the direct path from start to target is blocked by a rigid obstacle (black).

the PIIRL optimization problem. Since the active learning process involves some random selections, we ran 15 trials; 10 demonstrations were required in 14 cases, and 8 demonstrations in 1 case, with average recovered sensitivities (average [std.dev.]) being $L_1 = 0.100[0.0]$, $L_2 = 0.101[0.0002]$, $M_1 = 2.858[0.012]$, $M_2 = 8.630[0.071]$, $H_1 = 984.12[29.272]$, $H_2 = 999.509[0.165]$. Comparing these results with the sensitivities learned using the full set of demonstrations (see Section 4.4.1), we observe that the values are not meaningfully different, which shows that the active learning method can infer very similar sensitivity relationships with fewer demonstrations.

4.4.3 Physical Environment Tests



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

Figure 4.7: Testing for our physical test environment (a), with objects numbered and start (red) and goal (blue) states shown. Swept volumes of (b) path planned with uniform object sensitivity values, (c) demonstration path, and (d) path planned with recovered sensitivity values.

In addition to testing with our simulated environment, we have also applied our framework to a planar physical test environment shown in Figure 4.7 with an L-shaped block. Like our earlier experiments, the use of a planar 3DoF

environment allows for the deformation of objects in the environment to be tracked in real time by an overhead camera. Paths in the environment were planned using the same RRT* planner as before, albeit in $SE(2)$.

For comparison purposes, we first planned using uniform sensitivity values for all objects, as shown in Figure 4.7b. A demonstration path through a narrower, higher-deformation passage was provided using our demonstration capture environment, as shown in Figure 4.7c. As with the simulated environment, 100 samples were generated around the demonstration, and object sensitivity values $O_1 = 1.00$, $O_2 = 200.00$, $O_3 = 100.03$, $O_4 = 200.00$, $O_5 = 36.21$ were recovered using a lower bound of 1, upper bound of 200, and initial value of 100. These parameters are expected, as the demonstration path deforms O_1, O_4 and O_5, while avoiding the other objects. Planning using the recovered values is shown in Figure 4.7d; planning was performed with a planning time of 5 minutes. Following planning, all three paths were executed in our test environment by an industrial robot, with object deformations tracked by our tracking camera and reported in Table 4.2. As before, we do not expect the planned path to exactly match the demonstration; in particular due to the narrow low-cost passages in the environment, it is unsurprising that the planned path has significantly higher cost than the expert demonstration. However, the planned path does avoid O_3, instead preferring the passage between O_1 and O_2, which matches the preferences demonstrated by the expert.

4.5 Conclusion

We have developed a framework for recovering sensitivities of deformable objects so that our motion planners imitate the behavior of expert users in deformable environments. By formulating the problem of motion planning in deformable environments in terms of generating optimal paths that minimize deformation, we can recover object sensitivity parameters from demonstrated optimal paths using IOC. We also propose an active learning algorithm to generate demonstration tasks. Our framework has two advantages over existing similar techniques. First, by using sampling-based techniques for IOC that avoid the need to solve the forward problem and sampling-based asymptotically-optimal planners, our framework is more applicable to higher-dimensional problems than existing approaches. Second, our method for automatically generating demonstration tasks for users to perform reduces the number of demonstration tasks needed to capture the desired behavior. We tested our framework in simulated and physical test environments, and showed that it recovers object sensitivities suitable for planning paths that imitate the behavior of expert demonstrations. We also showed that these preferences can generalize to new tasks.

	Demonstrated			Recovered (30 min/plan)			Recovered (60 min/plan)		
	L	M	H	L	M	H	L	M	H
1	5.61	0.0	0.0	12.14 [4.68]	0.0 [0.0]	0.0 [0.0]	10.81 [1.55]	0.0 [0.0]	0.0 [0.0]
2	7.14	0.0	0.0	12.35 [2.7]	0.0 [0.0]	0.0 [0.0]	11.57 [2.82]	0.0 [0.0]	0.0 [0.0]
3	4.87	0.0	0.0	10.65 [3.71]	0.16 [0.82]	0.0 [0.0]	9.82 [2.84]	0.0 [0.0]	0.0 [0.0]
4	5.30	0.0	0.0	10.27 [2.75]	0.07 [0.38]	0.01 [0.03]	11.06 [2.22]	0.0 [0.0]	0.0 [0.0]
5	7.69	0.0	0.0	13.65 [4.18]	0.0 [0.0]	0.1 [0.53]	14.17 [3.62]	0.0 [0.0]	0.0 [0.0]
6	7.92	0.0	0.0	10.73 [2.09]	0.0 [0.0]	0.0 [0.01]	11.24 [4.7]	0.0 [0.0]	0.0 [0.0]
7	7.27	0.0	0.0	11.86 [2.93]	0.1 [0.54]	0.0 [0.0]	12.48 [3.5]	0.0 [0.0]	0.0 [0.0]
8	9.55	0.0	0.0	13.48 [3.52]	0.34 [1.47]	0.0 [0.0]	11.97 [2.97]	0.26 [0.97]	0.0 [0.0]
9	0.0	35.59	0.0	0.02 [0.13]	32.55 [9.13]	0.0 [0.01]	0.03 [0.11]	31.51 [10.48]	0.0 [0.0]
10	0.0	20.38	0.0	0.9 [1.63]	21.8 [8.44]	0.0 [0.01]	1.44 [2.69]	20.51 [8.5]	0.0 [0.0]
11	0.0	18.67	0.0	0.02 [0.08]	23.79 [5.62]	0.29 [1.29]	0.17 [0.56]	24.68 [5.55]	0.0 [0.0]
12	0.0	17.17	0.0	9.58 [1.95]	0.1 [0.32]	0.07 [0.25]	9.36 [1.96]	0.02 [0.09]	0.03 [0.09]

Table 4.1: Comparison between demonstrated behavior and paths planned using object sensitivity values recovered from 12 demonstrations between each pair of adjacent objects. Costs reported (mean [std.dev.]) are the integral of volume change multiplied by the true object sensitivity values, separated by class of object (L = low-sensitivity, including objects L_1 and L_2, M = medium-sensitivity, including objects M_1 and M_2, H = high-sensitivity, including objects H_1 and H_2).

	Object deformation				
	O_1	O_2	O_3	O_4	O_5
Uniform	0	3367	2442	0	554148
Demonstration	23451	0	0	0	35222
Recovered	51569	38798	0	0	73013

Table 4.2: Deformation comparison for the five left-hand objects in our physical test environment between a path planned with uniform object sensitivity values, the demonstrated path, and a path planned using the recovered sensitivity values. Reported deformation values are in pixels.

Chapter 5

Planning and Resilient Execution of Policies For Manipulation in Contact with Actuation Uncertainty

5.1 Introduction

Many real-world tasks are characterized by uncertainty: actuators and sensors may be noisy, and often the robot's environment is poorly modelled. Unlike robots, humans effortlessly perform everyday tasks, like inserting a key into a lock, which require fine manipulation despite limited sensing and imprecise actuation. We observe that humans often perform these tasks by exploiting *contact*, *compliance*, and *resilience*. Using compliance to safely make contact and move while in contact allows us to reduce uncertainty. We also exhibit resilience: when an action fails to produce the desired result, we may withdraw and try again. Seminal motion planning work by Lozano-Pérez et al. [42] shows that incorporating contact and compliance is critical to performing fine motions like inserting a peg into a hole. Building from this work and our observations of human motions, we have developed a motion planner that incorporates contact, compliance, and resilience to generate behavior for robots with actuation uncertainty.

Motion in the presence of actuation uncertainty is an example of a continuous Markov Decision Process (MDP), adding in sensor uncertainty, the problem becomes a Partially-Observable Markov Decision Process (POMDP). Solving an MDP or POMDP is often framed as the problem of computing an optimal policy π^* that maps each state to an action a that maximizes the expected reward (e.g., the probability of reaching the goal). We focus on motion planning with actuation uncertainty, and thus we frame the problem as an MDP. This MDP

formulation is representative of the challenges face by low-cost and compliant robots such as Baxter or Raven, which have accurate sensing but noisy actuators.

Instead of planning in the configuration or state-space of the robot, we represent the uncertainty of the state of the robot as a probability distribution over possible configurations, and plan in the space of these distributions—the *belief space*[1]. The computational expense of optimal motion planning leads us to adopt a thresholding approach from *conformant* planning [90]. Instead of attempting to find a global optimal policy, we seek to generate a *partial* policy that allows a robot with actuation uncertainty to move from start configuration q_{start} to reach goal q_{goal} within tolerance ϵ_{goal} with at least planning threshold P_{goal} probability. A partial policy, which maps a subset of possible states to actions rather than a global policy that maps all states to actions, simplifies the problem and is appropriate for the single-query planning problems we seek to solve..

The complexity of robot kinematics and dynamics preclude analytical modeling of compliance and contact for practical, high-dimensional problems, and thus we rely on the ability to forward simulate the state of the robot given a starting state and action. In the presence of uncertainty, individual actions may have multiple distinct outcomes: for example, when trying to insert a key into a lock, some attempts will succeed in inserting the key, while some will miss the keyhole. In advance of performing such an action, we cannot *select* between desired outcomes (as is assumed in [10]). However, we can *distinguish* between the outcomes after the action is executed. We directly incorporate this behavior into our planner using *splits* and *reversibility*. Splits are single actions that produce multiple distinct outcomes, which we distinguish between using a series of clustering algorithms. Reversibility is the ability of a specific action and outcome to be "undone" and return to the previous state, which allows the robot to attempt the action again. Of course, the planner may not accurately model the outcomes of every action, so we incorporate an online adaptation process to update the planned policy during execution to reflect the results of actions.

Our primary contributions are thus 1) incorporating contact and compliance into policy generation, thus allowing contacts that other planners would discard but that, in fact, can be used to reduce uncertainty; and 2) introducing resilience into policy execution and thus significantly increasing the probability of successfully completing the task. Our experiments with simulated test environments suggest that our planner efficiently generates policies to reliably perform motion for robots with actuation uncertainty. We apply our methods to problems in $SE(2)$, $SE(3)$, and a simulated Baxter robot ($\mathbb{R}^7$) and show performance improvements over simpler methods and the ability to recover from an unanticipated blockage.

[1]The term *belief* is borrowed from POMDP literature, which assumes that the state is partially-observable. Though we consider only MDPs, we nevertheless use "belief" as it is a convenient and widely-used term for a distribution over states.

5.2 Problem Statement

We consider the problem of planning motion for a *controlled compliant* robot R with configuration space $\mathcal{Q}$ in an environment with obstacles E. For given start (q_{start}) and goal (q_{goal}), we seek to produce motion which allows the robot to reach q_{goal} within tolerance ϵ_{goal} with at least P_{goal} probability.

The robot is assumed to have actuation uncertainty modelled by $q_{t+1} = q_t + (\Delta q_t + r_{\Delta q})$ in which the next configuration q_{t+1} is the result of the previous configuration q_t, control input Δq and actuation error $r_{\Delta q}$. We assume that a function $\mathbf{F}$, which models the probability distribution of the uncertainty, is available from which to sample $r_{\Delta q} \sim \mathbf{F}(\Delta q)$ for a given Δq. Due to this actuation uncertainty, when executing actions in our planner the result is a belief distribution b. The robot is compliant, meaning that for a motion from collision-free $q_{current}$ to colliding $q_{desired}$, the resulting configuration q_{result} will be in contact and the robot will not damage itself or the environment.

Since the motion of the robot is uncertain, a path τ that is a discrete sequence of configurations may not be robust to errors. Instead, we wish to produce a partial policy $\pi : \mathcal{Q}' \rightarrow A$ that maps $\mathcal{Q}' \subseteq \mathcal{Q}$ to actions A such that for a configuration $q \in \mathcal{Q}'$, the policy returns an action to perform. Even π may not always be robust to unexpected errors, therefore during execution we wish to detect actions that do not reach their expected results; i.e., when an action produces $q_{result} \notin \mathcal{Q}'$. In such an event, we wish to adapt $\mathcal{Q}'$ and π such that $q_{result} \in \mathcal{Q}'$ and continue attempting to complete the task.

5.3 Methods

We have developed a motion planner consisting of an anytime RRT-based global planner and a local planner that uses a kinematic simulator to model robot behavior. Together, they produce a set of solution paths S, where each solution $s \in S$ is a sequence of nodes $n_i = (b_i, a_i)$, in which b_i is the belief distribution for n_i and a_i is the action that produced b_i. Using this set of solution paths, we construct a single partial policy π. As π is queried during execution, we update the policy to reflect the "true" state observed during the execution process.

Because it is difficult to model the belief state in contact using a parametric distribution, we use a particle-based approach similar to [10] in which we represent the belief b_i of node n_i with a set of configurations $q_1, q_2, ..., q_n$ that are forward-simulated by the local planner. Like previous work [10], we expect that performing some actions will result in multiple qualitatively different states as illustrated in Figure 5.1 (e.g., in contact with an obstacle some particles will become stuck on the obstacle while others slide along the surface). These distinct parts of the belief state, which we refer to as *splits*, are distinguished in our planner by a series of clustering operations. To ensure that all actions are adequately modeled, a fixed number of particles $N_{particles}$ is used to simulate every action; since splits reduce the number of particles at a given state, a new set of particles must be resampled for these states to avoid particle starvation.

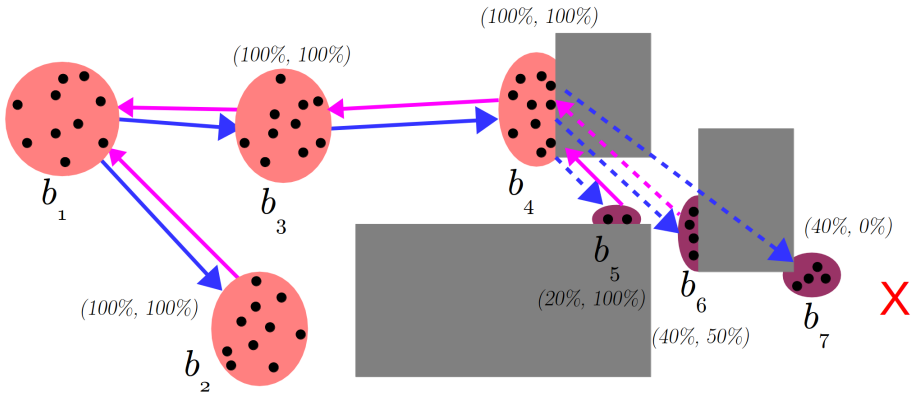


Figure 5.1: Our belief-space RRT extending toward a random target (red X) from b_4. Due to compliance, the particles (dots) can slide along the obstacles (gray). Solid blue edges denote 100% probability edges, dashed edges denote a split resulting in multiple states; solid magenta edges denote 100% reversible edges, while dashed edges denote lower reversibility. Because the extension is attempting to move through a narrow passage, particles separate and a split occurs, resulting in three distinct states (b_5, b_6, b_7).

It is important to understand that we cannot select between the different result states of a split when performing the action; however, we can *distinguish* using our clustering methods if we have reached an undesirable result. To be *resilient* to such errors, we incorporate the ability to reverse the action back to the previous state and try the action again. Clearly, not all actions will be reversible, so we perform additional simulation to estimate the ability to reverse each action after identifying the resulting states.

We first introduce our RRT-based global planner that uses a simulation-based local planner and a series of particle clustering methods to generate policies incorporating actuation uncertainty, and then discuss our online policy execution and adaptation that enables resiliency to unexpected behavior encountered during execution.

5.3.1 Global planner

Until it reaches time limit $t_{planning}$, our global planner iteratively grows a tree T using the local planner to extend the tree towards a sampled configuration q_{target}. Like the RRT, q_{target} is either a uniformly sampled $q_{rand} \in \mathcal{C}$, or with some probability, the goal q_{goal}. Each time we sample a q_{target}, we select the closest node in the tree $n_{near} = \text{argmin}_{n_i \in T} \textsc{Proximity}(b_i, q_{target})$. The local planner plans from n_{near} towards q_{target} and returns new nodes $\mathcal{N}_{new}$ and edges $\mathcal{E}_{new}$ that grow the tree. We check each new node $n_{new} \in \mathcal{N}_{new}$ to see if it meets the goal conditions, and if so, add the new solution path to S.

We also incorporate several features distinct from the RRT. First, using

PROXIMITY we consider more than distance when selecting the nearest neighbor node n_{near}. We want to bias the growth of the tree toward nodes that can be reached with higher probability and have more concentrated b_i, so we incorporate weighting using $P(n_{start} \rightarrow n_i)$, the probability the entire path from n_{start} to n_i succeeds, and $\text{var}(n_i)$, the variance of b_i. The proximity of a node n_i to a configuration q is given by the following equation:

$$\begin{aligned}\text{PROXIMITY}(n_i, q) &= \text{dist}(\text{expect}(b_i), q) \\ &* [(1 - P(n_{start} \rightarrow n_i)) * \alpha_P + (1 - \alpha_P)][\text{erf}(|\text{var}(b_i)|_1) * \alpha_V + (1 - \alpha_V)]\end{aligned} \tag{5.1}$$

Here, $\text{expect}(b_i) = q_{expected}$ is the expected value of the belief distribution b_i and $\text{dist}(q_{expected}, q)$ is the $\mathcal{C}$-space distance function. Two weights α_P and α_V control the effect of the probability and variance weighting, respectively. Values of α_P and α_V closer to 1 increase the effect of the weighting, while values closer to 0 increase the effect of the $\mathcal{C}$-space distance.Using the error function $\text{erf}(x) = {}^2/\sqrt{\pi} \int_0^x e^{-t^2} dt$ maps variance in the range $[0, \text{inf})$ to the range $[0, 1)$ to simplify computation. Previous work in belief-space planning has used a range of distance functions, such as L1, Kullback-Leibler divergence, Hausdorff distance, or Earth Mover's Distance (EMD) [69]; however, many of these choices only provide useful distances between belief states with overlapping support. While EMD encompasses both the C-space distance and probability mass of two beliefs, it is expensive to compute. Since most of our distance computations are between beliefs with non-overlapping support, the $\mathcal{C}$-space distance between expected configurations is an efficient approximation [69].

Second, we cannot simply test if $n_{new} = q_{goal}$, since the $P(n_{start} \rightarrow n_{new})$ may be low; instead, we check if a new solution has been found. To be a solution, the probability n_{new} reaches the goal must be greater than P_{goal}, i.e., the product of $P(n_{start} \rightarrow n_{new})$ and $|q \in b_{new} | \text{dist}(q, q_{goal}) \leq \epsilon_{goal}|$. Finally, once a path to the goal has been found, we continue planning to find alternative paths. We want to encourage a diverse range of solutions, so once a solution path has been found, we remove nodes on solution branches from consideration for nearest neighbor lookups. This process recurses towards the root of the planner's tree T until it either reaches the root node n_{start} or a node n_i which is the result of a split. Once the base of the solution branch is found, we remove the branch from nearest neighbors consideration and continue planning until reaching $t_{planning}$.

5.3.2 Local planner

Our local planner grows the planner tree T from nearest neighbor node n_{near} towards a target configuration q_{target} by forward-propagating belief using EXTEND to produce one or more result nodes $n_{new} \in \mathcal{N}_{new}$ and edges $e_{new} \in \mathcal{E}_{new}$ (recall that splits may occur). To improve the time-to-first-solution, the local

planner operates like RRT-Connect, repeatedly calling EXTEND, until a solution is found, whereupon it switches to RRT-Extend, calling EXTEND only once, to improve coverage of the space and encourage solution diversity. Note that the RRT-Connect behavior is stopped if an extension results in a split.

EXTEND forward-simulates particles $Q_{initial}$ from node n_{near} towards q_{target}, clusters the resulting particles $Q_{results}$ into new nodes $\mathcal{N}_{new}$, and computes the transition probabilities. As previously discussed, we simulate every action with the same number of particles. If node n_{near} is *not* the result of a split, and thus b_{near} contains a full set of particles, then we simply copy b_{near} to use for simulation. If, instead, n_{near} is the result of a split, then we uniformly resample $N_{particles}$ particles from b_i. We then simulate the extension toward q_{target} for each particle. Any simulation engine that simulates contact and compliance could be used, but the simulation should be as fast as possible to minimize planning time. In our experiments, we used an approximate kinematic simulator described in Appendix B. The resulting particles are then grouped into one or more clusters using CLUSTERPARTICLES, which we describe in Section 5.3.3. For each cluster $Q_{cluster}$, we form a new node $n_{new} = (b_{new}, a_{new})$ with belief $b_{new} = Q_{cluster}$ and action $a_{new} = q_{target}$. In the case of splits, where multiple nodes are formed, we assign $P(n_{near} \rightarrow n_{new,i}) = |b_{new,i}|/N_{particles}$. We then estimate the probability that action a_{new} can be reversed from node n_{new} by simulating $N_{particles}$ particles back towards node n_{near}. Note that some particles may become stuck while reversing, and thus the probability of reversing the action may not be 1.

The ability to reverse an action allows us to detect an undesired outcome, reverse to the parent node, and retry the action until we either reach the desired outcome or become stuck. Thus, we estimate the *effective* probability $P(n_{near} \rightarrow n_{new})_{effective}$ for each node n_{new} by estimating the probability that a particle has reached n_{new} after $N_{attempt}$ attempts, where at each attempt, particles that have not reached the n_{new} try to return to n_{near} and try again.

Analysis – The planner always stores $N_{stored} = N_{actions}N_{particles}$ particles. For every action $N_{particles}$ particles are forward-simulated, and all of them are stored in $\mathcal{N}_{new}$. In the worst case, where every action produces $N_{particles}$ distinct nodes, the number of particles that must be simulated $N_{simulated} = N_{nodes}(N_{particles} + 1)$, as each node itself is the product of one initial simulation and $N_{particles}$ simulations are required to estimate reverse probability. In practice, as we discuss in Section 5.4.1, the space requirements to perform complex tasks are low as most actions produce a small number of nodes, and the time cost can be reduced by simulating particles in parallel.

5.3.3 Particle clustering

Intuitively, we want every configuration in a cluster to be reachable from every other one using the local planner. However, testing this directly is computationally expensive, so we also consider two approximate methods. All clustering methods use two successive passes to cluster the configurations resulting from forward simulation: first, a spatial-feature-based pass that groups configurations

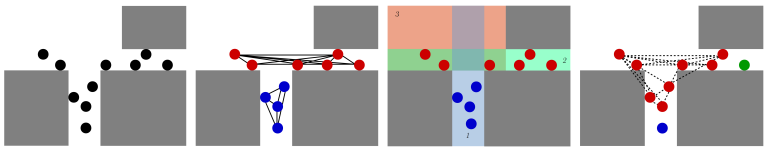


Figure 5.2: Our proposed spatial-feature particle clustering methods. (a) The positions of particles after an extension of the planner. (b) Actuation center clustering, with clusters (red, blue) and the straight-line paths for each cluster. (c) Weakly Convex Region Signature clustering, with the three convex regions shown and labeled. (d) Particle movement clustering, with successful particle-to-particle motions shown dashed for the main cluster (red) and two unconnected particles (blue, green).

based on their relationship to different parts of the workspace, and second, a distance-based pass that refines the initial clusters. All of our clustering methods use complete-link hierarchical clustering, as it produces smaller, more dense clusters, while not requiring the number of clusters to be known in advance [91, 10]. Below we discuss the ideal approach and our two approximations, shown in Figure 5.2. We compare the performance of these methods in Section 5.4.1.

Particle Connectivity (PC) Clustering

We run the local planner from every configuration to every other configuration and record which simulations reach within ϵ_{goal} of the target. For a pair of configurations q_1, q_2, going from q_1 to q_2 may fail while the opposite succeeds; however, to be conservative, we only record success if both executions succeed. We then perform clustering using the complete-link clustering method with distance threshold 0, where successful simulations correspond to distance 0 and unsuccessful simulations correspond to distance 1. Note that this method is very expensive, since it requires simulating $N^2 - N$ particles for N configurations considered.

Weakly Convex Region Signature (WCR) Clustering

Intuitively, in many environments a robot can move freely from q_1 to q_2 if both configurations reside entirely in the same convex region of the workspace. This is also true for some slight concave features, so long as the features do not block the robot. Conversely, for configurations in clearly distinct regions, it is less likely that the robot can move from one configuration to the other.

Illustrated in Figure 5.2c, we capture this intuition by recording the position of the robot relative to *weakly convex regions* of the free workspace, to form what we call the *convex region signature*. These regions form a weakly convex covering: individual regions may contain slight concavity, and multiple regions overlap. Techniques such as [92] exist to automatically compute these regions,

but for simple environments these regions can be directly encoded. The convex region signature of a configuration q, $WCR(q)$, records the region(s) occupied by every point of the robot at q. Distance metric D_{WCR} between two region signatures $WCR(q_1)$ and $WCR(q_2)$ is the percentage of points in the robot that do *not* share a common region between the signatures. Using this metric, we perform complete-link clustering. We test different thresholds for D_{WCR} in Section 5.4.1. This method allows configurations with some points in a shared region to be clustered together, while separating configurations that share no regions. At runtime, this method requires N computations of $WCR(q)$ and $^{(N^2-N)}/_2$ evaluations of D_{WCR} to compute all pairwise distances. An alternative to this method would be the use of Reeb graph representations which recent motion planning work suggests is useful to capture topological information [93].

Actuation Center (AC) clustering

We observe that many successful motions in contact occur when the actuation (or joint) centers of the starting and ending configuration can be connected by collision-free straight lines, so this method checks the straight-line path from the joint centers of one configuration to those of the other configuration. As with the particle movement clustering approach, configurations with successful (collision-free) paths have distance 0, while those with unsuccessful (colliding) paths have distance 1. Like the previous approach, clusters are then produced using complete-link clustering with threshold 0. At runtime, this method requires $^{(N^2-N)}/_2$ checks of the straight-line paths.

5.3.4 Partial policy construction

Once the global planner has produced a set of solution paths S, we construct a partial policy π. Policy construction consists of the following steps:

1. Graph construction – An explicit graph is formed, in which the vertices of the graph are nodes $n_i \in S$, and the edges correspond to the edges forming the paths in S. An edge $n_i \rightarrow n_{i+1}$ is assigned an initial cost $1/P(n_i \rightarrow n_{i+1})$). This means that likely edges receive low cost, which is necessary to compute maximum-probability paths through the graph.

2. Edge cost updating – The edge costs are updated to reflect the estimated number of attempts needed to successfully traverse the edge by multiplying the cost of the edge by the estimated number of attempts required to reach $P(n_i \rightarrow n_{i+1}) \geq P_{goal}$. This estimate is the complement of the effective probability discussed in Section 5.3.2; instead of computing the probability of reaching a node after a fixed number of attempts, we compute the number of attempts needed to reach the node with P_{goal} probability. The fewer attempts necessary to traverse the edge, the faster the policy can be executed, and thus this cost represents an expected execution time.

3. Dijkstra's search – The optimal path from every vertex in the graph to the goal state is computed using Dijkstra's algorithm. This determines

Algorithm 4 Partial policy query algorithm

procedure PolicyQuery($S, \pi, q_{current}, a_{performed}$)
  $\mathcal{N}_{potential} \leftarrow \{n_i \in S \mid a_i = a_{performed}\}$
  $\mathcal{N}_{matching} \leftarrow \{n_i \in \mathcal{N}_{potential} \mid |ClusterParticles(b_i \cup q_{current})| = 1\}$
  if $\mathcal{N}_{matching} \neq \emptyset$ **then**
    $n_{reached} \leftarrow argmin_{n_i \in N_{matching}}$DijkstraDistance($n_i$)
    IncreaseProbability($n_{reached}, a_{performed}$);
    for $n_i \in \mathcal{N}_{potential} \mid n_i \neq n_{reached}$ **do**
      ReduceProbability($n_i, a_{performed}$)
    $\pi \leftarrow$ ConstructPolicy(π)
    if $P(n_{reached} \rightarrow q_{goal}) \geq P_{goal}$ **then**
      $a_{next} \leftarrow \pi(n_{reached})$
      return a_{next}
    else
      return failure
  else
    $n_{observed} \leftarrow \{\{q_{current}\}, a_{performed}\}$
    $S \leftarrow S \cup n_{observed}$
    return PolicyQuery($S, \pi, q_{current}, a_{performed}$)

the optimal next state (and thus action to perform) for every state in the graph.

5.3.5 Partial policy execution and adaptation

At every step during execution, the partial policy π is queried for the next action to perform. While we could simply find the "closest" node in the policy using a distance function like Equation 5.1, doing so would discard important information. Not only do we know the configuration $q_{current}$ that results from executing an action, but we also know the action $a_{performed}$ we attempted to perform. Using this information, we know exactly which nodes(s) in π the robot should have reached. As shown in Algorithm 4, we first collect all potential result nodes (i.e., those nodes n_i with actions $a_i = a_{performed}$). We then use our particle clustering method to cluster $q_{current}$ with the belief b_i of each n_i. This clustering tells us if the robot reached a given state (if a single cluster is formed) or not (multiple clusters). In the unlikely (but possible) event that $q_{current}$ clusters with multiple potential result nodes, we select the "best" matching node $n_{reached}$ using the distance-to-goal computed via Dijkstra's algorithm.

The key contribution of our policy execution is that we adapt the policy π to reflect the results of actual execution. If a matching node $n_{reached}$ is found, we then update π to increase the probability that $n_{reached}$ is the result of $a_{performed}$. We assign a constant $A_{importance} \in \mathbb{N}$ that reflects how much we value the results of executing an action compared to the results of simulating a particle during planning. To update the probability, we increase the counts of

attempted $N_{attempts}$ actions and successful $N_{successful}$ actions, then recompute probability:

$$P(n_{previous} \rightarrow n_{reached}|a) = \frac{N_{successful} + A_{importance}}{N_{attempts} + A_{importance}} \tag{5.2}$$

Likewise, we reduce the probability for other potential result states:

$$P(n_{previous} \rightarrow n_{other}|a) = \frac{N_{successful}}{N_{attempts} + A_{importance}} \tag{5.3}$$

This update process allows us to learn online, during execution, the true probabilities of reaching states given an action. In effect, the probabilities computed by the global planner serve as an initialization for this online learning. Once updated, we rebuild policy π to reflect the new probabilities. If the probability of reaching the goal $P(n_{reached} \rightarrow q_{goal})$ is at least P_{goal}, we query π for the next action to take. If the probability of reaching the goal has dropped below P_{goal}, policy execution terminates.

However, sometimes no matching node $n_{reached}$ exists. This means a split occurred during execution that was not captured in S during planning (e.g., an obstacle that is not accurately modelled in E, or where the behavior of the simulator diverges from the true robot). To handle this case, we insert a new node $n_{observed}$ with belief $b_{observed} = \{q_{current}\}$ into S, and then retry the policy query (which will now have an exactly matching state). To incorporate reversibility, we initially assign new nodes a reverse probability $N_{attempts} = N_{successful} = 1$. Thus, the next action selected by the policy will be to return to the previous node. Together with updating probabilities by inserting new states in this manner, we can thus extend the policy to reflect behavior observed during execution that was not captured during the planning process.

Analysis – In the worst case, a policy π cannot be executed successfully, and performing every action a results in a new node $n_{observed}$. For any $A_{importance} \in \mathbb{N}, P_{goal} > 0$, adapting the policy will detect failure and terminate in this case.

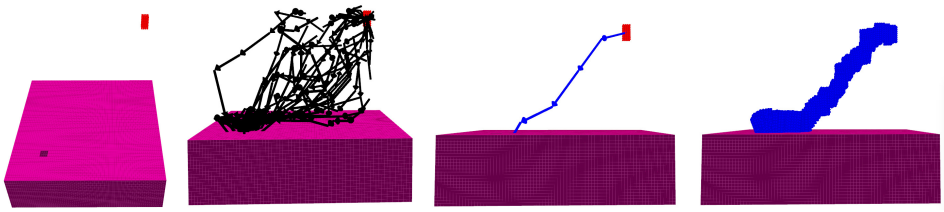
Proof – For every action $a_{i+1}, ...,$ node $n_{observed}$ will be created with a reverse prior $P(n_{observed} \rightarrow n_{previous}) = 1/1$. If reversing to $n_{previous}$ fails, we update $P(n_{observed} \rightarrow n_{previous}) = 1/1+A_{importance}$. For the ith successive failed reverse and $n_{observed,i}$ generated, $P(n_{observed,i} \rightarrow n_{previous}) = \Pi_i \frac{1}{1+A_{importance}}$. As the number of failed actions increases $P(n_{observed,i} \rightarrow n_{previous}) \rightarrow 0$, and thus $P(n_{observed,i} \rightarrow q_{goal}) \leq P(n_{observed,i} \rightarrow n_{previous}) \rightarrow 0$. Thus eventually $P(n_{observed,i} \rightarrow q_{goal})$ will fall below $P_{goal} > 0$ and execution will terminate. □

5.4 Results

We present results of testing our planner in simulated $SE(2)$ and $SE(3)$ environments and a simulated R^7 Baxter robot. For dynamic simulation during execution, we use the Gazebo simulator [94]. As our kinematic simulator does

not consider friction, we use *contact motion controllers* to reduce contact forces (see Appendix C). We present statistical results over a range of actuation uncertainty and clustering methods and show that our planner produces policies that allow execution of tasks incorporating contact and robot compliance. We also present statistical results showing that our online policy updating adapts to unexpected behavior during execution. All planning and simulation testing was performed using 2.4 GHz Xeon E5-2673v3 processors. Likewise, all planning was performed with PROXIMITY weights $\alpha_P = \alpha_V = 0.75$ (see Equation 5.1), $N_{attempt} = 50$ attempted reverse/repeats of each action, and planning threshold $P_{goal} = 0.51$, such that solutions must be more likely than not to reach the goal.

5.4.1 $SE(3)$ simulation



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

Figure 5.3: (a) The $SE(3)PegInHole$ task involves moving from the start (red) to the bottom of the hole. (b) An example policy produced from 296 solutions, the (c) initial action sequence (blue arrows), actions the policy will return if every action is successful, and (d) the swept volume of the peg executing the policy. Note that the peg makes contact with the environment to reduce uncertainty, then slides into the hole.

	AC	PC	WCR with $D_{WCR} =$				
			0.125	*0.25*	*0.5*	*0.75*	*0.99*
P_{plan}	1.0	1.0	1.0	0.97	0.97	0.97	0.97
P_{exec}	0.97	0.89	0.73	0.95	0.84	0.99	0.96
	[0.17]	[0.19]	[0.42]	[0.18]	[0.34]	[0.02]	[0.18]

Table 5.1: $SE(3)PegInHole$ particle clustering performance comparison (mean [std.dev.]) of P_{exec}, the probability of reaching the goal with 300 seconds, between policies produced using our planner with different clustering methods. P_{plan} is the probability that a policy is planned within 5 minutes, averaged over 30 plans, and P_{exec} is averaged over 40 executions on each successfully-planned policy.

Peg-in-hole

In $SE(3)PegInHole$, a version of the classical peg-in-hole task [42] shown in Figure 5.3, the free-flying 6-DoF robot "peg" must reach the bottom of the hole.

	Simplified				Planned policies (WCR, $D_{WCR} = 0.75$)			
	Simple RRT		Contact RRT		24 particles		48 particles	
γ	P_{plan}	P_{exec}	P_{plan}	P_{exec}	P_{plan}	P_{exec}	P_{plan}	P_{exec}
0	0	0 [0]	1	0.78 [0.38]	1	0.42 [0.48]	0.97	0.59 [0.47]
1/16	0	0 [0]	1	0.78 [0.39]	1	0.60 [0.43]	1	0.625 [0.43]
1/8	0	0 [0]	1	0.79 [0.38]	1	0.99 [0.18]	0.93	0.81 [0.37]
1/4	0	0 [0]	1	0.50 [0.37]	1	0.90 [0.28]	0.86	0.72 [0.41]

Table 5.2: $SE(3)PegInHole$ policy performance comparison between simplified planners and our planner with 24 and 48 particles and actuation uncertainty γ.

This task is difficult for robots with actuation uncertainty, as the hole is only 30% wider than the peg. Even without uncertainty, attempting to avoid contact greatly restricts the motion of the robot entering the hole. Instead, as shown in [42], the best strategy is to use contact with the environment and the compliance of the robot to guide the peg into the hole. We assess the performance of a policy approach in terms of P_{exec}, the probability that executing the policy reaches the goal within a time limit of 300 seconds. For a given value of γ, linear velocity uncertainty $\gamma_v = \gamma$ (m/s) and angular velocity uncertainty $\gamma_\omega = 1/4\gamma$ (rad/s). Linear and angular velocity noise is sample from a zero-mean truncated normal distribution with bounds $[-\gamma_{v,\omega}, \gamma_{v,\omega}]$ and standard deviation $1/2\gamma_{v,\omega}$. While this differs from zero-mean normal distributions conventionally used to model uncertainty, we believe the bounded truncated distribution better reflects the reality of robot actuators, which do not exhibit unbounded velocity error. Goal distance threshold ϵ_{goal} was set to 1/2 the length of the peg.

We first compared the performance of our planner at a fixed $\gamma = 1/8$ and $N_{particles} = 24$ using the clustering approaches introduced in Section 5.3.3, including several thresholds for $D_{WCR} = 0.125, 0.25, 0.5, 0.75, 0.99$, with 30 plans per approach (5 minutes planning time) and 40 executions of each planned policy. As seen in Table 5.1, WCR clustering with $D_{WCR} = 0.75$ clearly outperformed the others in terms of policy success, reaching the goal in 99% of executions. Planning time is overwhelmingly dominated by simulation, accounting for approximately 99.9% of the allotted time. Using WCR and $D_{WCR} = 0.75$, we then compared the performance of our planner against two simplified RRT-based approaches:

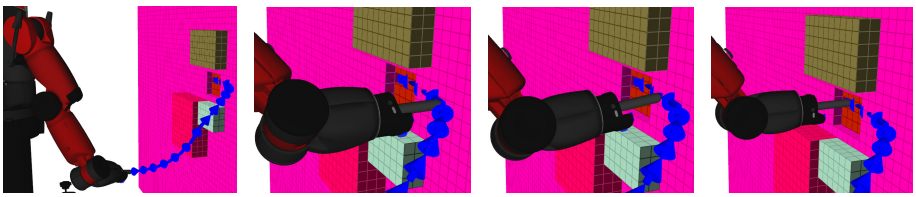
1. Simple RRT – Does not model uncertainty or allow contact, but like our planner produces multiple solutions in the allotted planning time.

2. Contact RRT – Incorporates contact and compliance but does not model uncertainty. Equivalent to planning with $\gamma = 0$ and one particle.

In addition, we tested our planner with both 24 and 48 particles to show

the effects of increasing the number of particles used. As before, we planned 30 policies for each, and executed each planned policy 40 times. Note that the Simple RRT was unable to produce solutions in 5 minutes due to the confined narrow passage. Results are shown in Table 5.2. With low actuator error the Contact RRT performs better, as it does not expend planning time on simulating multiple particles and instead produces more solutions. As error increases, our planner clearly outperforms the alternatives. Note that increasing particles does not improve performance, indicating that 24 particles is sufficient without requiring unnecessary simulation. Low P_{exec} overall, in particular when planning with low γ, is due to the mismatch between the planning simulator and the dynamics of Gazebo (i.e., motions that are possible in the planner, but not in Gazebo) which disproportionally affects motions near the entrance to the hole. In particular, the planner at low values of γ overestimates how successfully motions at the edge of the hole can be performed and thus results in a lower-than-expected P_{exec}.

In terms of the number of particles stored, the worst case was $N_{particles} = 48, \gamma = 0$, with an average of 148894 (std.dev. 25015) particles stored. The worst case for simulated particles was $N_{particles} = 24, \gamma = 1/4$, with an average 268634 (std.dev. 16856) particles simulated. In practice, the storage and computational expense is limited; the worst-case for particles stored requires a mere 15 megabytes, while for a planning time of 300 seconds and using eight threads, the planner evaluated more than 100 particles per second per thread.



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

Figure 5.4: An execution of the Baxter task, from start (a) to goal (d), and environment with confined space around the goal. The planned policy is shown in blue. Note the use of contact with the environment to reduce uncertainty and reach the target passage.

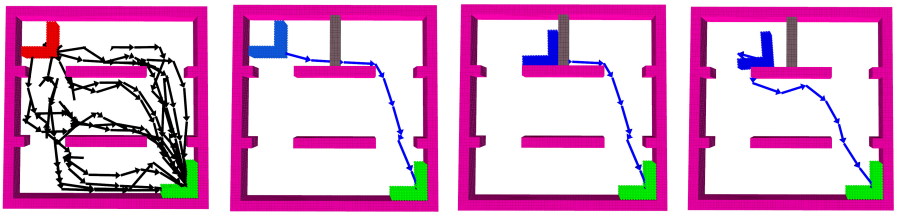
5.4.2 Baxter simulation

In addition to $SE(3)$ and $SE(2)$ tests, we apply our planner and policy execution to a simulated Baxter robot shown in Figure 5.4, with the robot reaching into a confined space. We compare the performance of our planner with $N_{particles} = 24$ and WCR clustering method with $D_{WCR} = 0.1$ with the simplified Contact RRT in terms of success probability P_{exec} for uncertainty $\gamma = 0.1$. To simulate Baxter's actuation uncertainty γ defines a truncated normal distribution with $\sigma = 1/2\gamma\dot{q}_i$ and bounds $[-\gamma\dot{q}_i, \gamma\dot{q}_i]$ for each joint i with velocity $\dot{q}_i$. Goal distance

threshold $\epsilon_{goal} = 0.15$ radians. We generated 10 policies using each approach with a planning time of 10 minutes to ensure both approaches would produce multiple solutions, then executed each 8 times for up to 5 minutes. As expected, Contact RRT finds solutions faster; on average 8.42s (std.dev. 2.61) versus 65.4s (std.dev. 32.9) and policies contain more solutions; on average 17.2 (std.dev. 16.5) versus 6.33 (std.dev. 3.97) since each solution requires less simulation time. However, our planner incorporating uncertainty outperforms the Contact RRT baseline with $P_{exec} = 0.79$ (std.dev. 0.30) versus $P_{exec} = 0.70$ (std.dev. 0.30). This suggests that, while planning with uncertainty does help in this environment, our approach to policy execution and resilience also works well when uncertainty is not accounted for in the planner, but we have a diverse policy.

5.4.3 Policy adaptation

We performed tests in a planar $SE(2)$ (3-DoF) environment to show that our policy adaptation recovers from unexpected behavior during execution. As shown in Figure 5.5, the L-shaped robot attempts to move from the start (upper left) to the goal (lower right). Due to the obstacles present, there are three distinct horizontal passages. Using the same controllers and uncertainty models as the $SE(3)$ tests with uncertainty $\gamma = 0.125$ and WCR clustering method with $D_{WCR} = 0.75$, 24 particles, and a planning time of 2 minutes, we generated 30 policies using our planner. Goal distance threshold ϵ_{goal} was set to $1/8$ the length of the robot.



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

Figure 5.5: (a) Our planar test environment, in which the robot must move from upper left (red) to lower right (green), with an example policy produced by our planner, with solutions through each of the horizontal passages. (b) The initial action sequence (blue arrows), showing actions the policy will return if every action is successful. (c) Following the policy, the robot becomes stuck on the new obstacle (gray). (d) Once the policy detects the failed action, it adapts to avoid the obstacle.

We evaluated the performance of the planned policies in the unmodified environment and an environment in which we blocked the horizontal passage used by the initial path of each policy. Each was executed 8 times for a total of 240 policy executions, for a maximum of 600 seconds. In the unmodified environment, 97% of policies were executed successfully, with an average of 15.4 actions

(std.dev. 9.62). In the modified environment with policy execution importance $A_{importance} = 500$ (this high value results in rapid policy adaptation), 73% of policies were executed successfully, with an average of 26.7 actions (std.dev. 12.3). This result shows that adapting the policy using our methods allows us to circumvent the new obstacle, however, doing so may result in following a path that is less likely to succeed.

5.4.4 Discussion

Our planner relies on our kinematic simulator to model compliant robot behavior. While this is an implementation choice and a more accurate simulator could be used, the advantage of our simulator is that it is fast enough to allow useful planning times. Our kinematic simulator offers a computationally efficient method for simulating contact and compliance; however, since our simulator does not model dynamic behavior such as inertia, forces, torques, or friction directly, it cannot capture behavior that depends on them. In particular, this occurs when simulating multi-point contact and multi-link robots. To bridge the gap between our approximate simulator and real dynamic simulation, we use *contact motion controllers* that serve to reduce interaction forces and allow the real robot to better follow the planned motions.

In straightforward cases of sliding contact along one or more surfaces, the combination of our simulator and controllers perform well. However, as is common to many simulators, multi-point contact can be challenging. This can occur as part of "jamming" behavior where the robot becomes stuck against the environment as the result of multiple contacts blocking the motion of the robot. Motion in these configurations is highly dependent on interaction forces, and thus our simulator does not accurately model them. This may be mitigated in certain cases by using the simulator in a model-predictive manner where the approximate simulator evaluates an idealized compliant configuration which is then used to adjust the target of the robot's motion. However, in our experience, jamming behavior is not reliably handled by our controllers: if reducing interaction forces is not enough to "unjam" the robot, the motion will fail.

Compliant behavior of a multi-link robot in contact with the environment depends on the exact torques of actuators and magnitudes of interaction forces. In our testing with the Baxter robot, the contacting configurations simulated by our simulator did not match the configurations produced by simulating in Gazebo or by executing on the real robot. This does not mean our simulator produces *invalid* configurations; rather, it produces *a* possible complied configuration. This limitation of our simulator can be mitigated by using the simulator in a model-predictive manner. In this manner, we can effectively adapt the real robot to match the limitations of the simulator. A similar challenge is posed when trying to resolve self-collisions between links of a multi-link robot. In our limited experience, the self-contacting configurations returned by our simulator do not exactly match those encountered on a real robot. Here, again, the use of our simulator in a model-predictive controller allows real robot behavior to be "idealized" to match the simulator.

5.5 Conclusion

We have developed a method for planning motion for robots with actuation uncertainty that incorporates environment contact and compliance of the robot to reliably perform manipulation tasks. First, we generate partial policies using an RRT-based motion planner that uses particle-based models of uncertainty and kinematic simulation of contact and compliance. Second, we adapt planned policies online during execution to account for unexpected behavior that arises from model or environment inaccuracy. We have tested our planner and policy execution in simulated $SE(2)$ and $SE(3)$ environments and on the simulated Baxter robot and show that our methods generate policies that perform manipulation tasks involving significant contact and compare against two simpler methods. Additionally, we show that our policy adaptation is resilient to significant changes during execution; e.g., adding a new obstacle to the environment.

Acknowledgements

This work was supported in part by NSF grants IIS-1656101 and IIS-1551219.

Chapter 6

Towards an Efficient Path Quality Metric For Compliant Robots with Actuation Uncertainty

6.1 Introduction

In recent years, robotics researchers and developers have introduced a number of compliant robots aimed at a range of tasks from human-robot collaboration to factory assembly. Compliance improves the safety of both the robot and its surroundings, as the robot can safely make unexpected contact without damage to itself or the surroundings. However, this compliance often comes at a cost of actuator accuracy, in particular on robots where compliance is implemented by means of passive hardware. As a result, these robots often exhibit significant actuation uncertainty. In practice, this means that executing collision-free planned plans can result in unexpected contact with the environment. While this contact is safe for a compliant robot, it may prevent the robot from successfully completing the planned path. Thus, we seek to develop an efficient path quality metric that reflects the robustness of a path to actuation uncertainty during execution; i.e., a metric that approximates the probability a robot executing the path successfully reaches the goal.

The availability of such a path quality metric has important applications in motion planning with uncertainty. While existing approaches to motion planning with uncertainty like that presented in Chapter 5 are able to incrementally estimate the probability of successfully executing a path during the planning process, this often requires significant expensive simulation for each state considered by the planner. Notably, in our previous experience, even in tasks with significant actuation uncertainty, planners that ignored actuation uncertainty

remained competitive with planners that modelled uncertainty as the simpler planners were able to explore significantly more candidate states and produce more robust and diverse execution policies. This experience suggests that many tasks may be successfully planned without accounting for uncertainty, but ensuring that these planners produce paths with sufficient quality requires a quality metric. Potentially, such a metric can be used to distinguish between simple tasks that can be efficiently planned using simpler planners and "uncertainty-sensitive" tasks that require more powerful planners. An additional role of such a metric is as an objective function in a trajectory optimizer, in which the quality metric can be used to optimize the path to improve robustness.

A key component to developing this metric is modelling the compliant behavior of the robot and environment; however, the complexity of real high-dimensional robot kinematics and dynamics preclude the development of a useful analytical model. Instead, as with our previous work, we rely on the availability of forward simulation given a starting configuration and desired action. Of course, such a simulator provides a naïve metric for path quality: simply by simulating a large number of executions of the path, we may estimate the probability the robot reaches the goal or instead becomes stuck. Such a naïve metric, however, would be extremely expensive and slow to compute for non-trivial numbers of simulated executions. Achieving high coverage of the $\mathcal{C}$-space volume surrounding the path, in particular near the contact manifold with the environment, requires a high number of executions. Similarly, such a solution would face a problem common to particle filters, in which a large number of "particles" (i.e., simulated robots) become stuck in confined parts of the environment and thus the simulated robots would need to be "resampled" to ensure converge throughout the path. However, it is very unclear how such simulated robots could be consistently resampled. Instead, we seek to develop a metric that is both efficient to compute and ensures coverage of the path without requiring superfluous simulation.

We develop two variants of a path quality metric. In both variants, we first compute the reachable $\mathcal{C}$-space volume between each pair of adjacent waypoints in the path. Within this reachable volume, we sample configurations and identify those close to the contact manifold. For these configurations, we use forward simulation to determine if they collide with the environment and become stuck. In the first, primary variant, we use the images of the sampled configurations to directly estimate the probability that a robot executing the path becomes stuck in each reachable volume, from which we can compute the probability of successfully reaching the next waypoint. In the simplified second variant, we use stuck configurations to identify where in the path and how many simulated executions to perform, and then use simulation to estimate the probability of successfully moving to the next waypoint. By computing these probabilities for every adjacent waypoint-waypoint pair, we can then compute the probability of successfully executing the path. We extend the work of [95] in computing reachable volumes for kinematic linkages to *trajectory*-linkages so that we can compute the reachable volume of $\mathcal{C}$-space surrounding the path. The use of images and pre-images in robot motion with uncertainty has been long established,

with the seminal work of Lozano-Pérez et al. [42], however, it has been shown that computing pre-images exactly is expensive [43, 45]. Instead of attempting to exactly compute them, we propose approximating the images using sampling.

6.2 Problem Statement

Specifically, we seek to develop a path quality metric $\mathbf{M} : \Pi \to \mathbb{R}$ that computes the quality of an arbitrary path Π composed of waypoints $\pi_{start}, \pi_1, \pi_2, ..., \pi_{goal}$ executed by a *controlled compliant* robot R with configuration space $\mathcal{Q}$ and configuration q in an environment with obstacles E. For a robot with target waypoint π_i, we step forward to the next waypoint $\pi_j, j > i$ only after $\text{dist}(q, \pi_i) \leq \epsilon_{reached}$. We similarly define the quality of a path Π as $P_{exec}(\Pi)$, the probability that a robot executing the path reaches π_{goal} within tolerance $\epsilon_{reached}$.

The robot is assumed to have actuation uncertainty modelled by $q_{t+1} = q_t + (\Delta q_t + r_{\Delta q})$ in which the next configuration q_{t+1} is the result of the previous configuration q_t, control input Δq and actuation error $r_{\Delta q}$. We assume that a function $\mathbf{F}$, which models the probability distribution of the uncertainty, is available from which to sample $r_{\Delta q} \sim \mathbf{F}(\Delta q)$ for a given Δq. The robot is compliant, meaning that for a motion from collision-free $q_{current}$ to colliding $q_{desired}$, the resulting configuration q_{result} will be in contact and the robot will not damage itself or the environment.

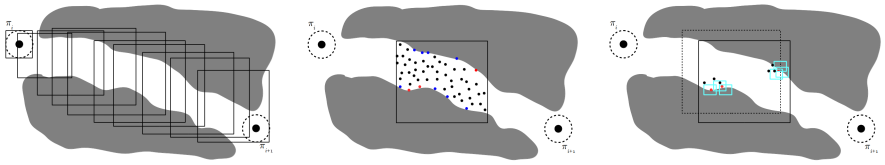
6.3 Trajectories as Reachable Volumes

As a consequence of the manner we select target waypoints, we ensure that a robot moving from waypoint π_i to waypoint π_j must start within $\epsilon_{reached}$ of π_i. Therefore, we can consider each pair of adjacent waypoints independently, and thus $P_{exec}(\Pi) = P_{exec}(\pi_1 \to \pi_{goal})$, the probability the entire path Π is executed successfully, is the product of all $P_{exec}(\pi_i \to \pi_{i+1})$ where π_i and π_{i+1} are adjacent waypoints in Π. Thus, our problem is to compute $P_{exec}(\pi_i \to \pi_{i+1})$ for an arbitrary pair of adjacent waypoints.

Motion between adjacent waypoints defines a volume of $\mathcal{C} - space$ covering all possible robot configurations in the process of moving from one waypoint to the next. Within this volume, we seek to identify regions where the robot may become stuck and compute the likelyhood of the robot passing through these regions. Computing a continuous representation of this $\mathcal{C} - space$ volume is non-trivial; however, since the robot is controlled, we can use the timestep Δt of controller cycles to discretize the volume into a set of $\mathcal{C} - space$ volumes corresponding to points in time, which we refer to as *reachable volumes*. Of course, computing these volumes exactly in the presence of obstacles is intractable; doing so would require extensive simulation to model possible robot trajectories. Since we have assumed the provided path is visible, we can make the conservative approximation that the robot's motion will follow the provided path, and

that *approximate reachable volumes* computed without accounting for obstacles can be used. Note that this approximation is conservative - by ignoring potential robot motions that leave these volumes due to contact (and potentially still reach the target waypoint) it can only underestimate the probability of successfully completing the motion.

6.4 Methods



(a) (b) (c)

Figure 6.1: (a) Reachable volumes are computed between starting waypoint π_i and ending waypoint π_{i+1}. (b) Configurations are randomly sampled within each reachable volume; colliding samples are omitted for clarity. Near-contact samples are projected to the contact manifold, then simulated to distinguish between *sliding* (blue) and *stuck* (red). (c) We estimate the probability that samples in the previous reachable volume (dashed box, select samples shown) lead to *stuck regions* defined by these stuck configurations. We integrate the image (cyan box) of each sample over these stuck regions to estimate the probability these samples become stuck.

While approximate reachable volumes provide a critical simplification of the $\mathcal{C}-space$ motion of the robot, the issue of capturing $\mathcal{C}-space$ configurations of the robot remains. Since the robots we wish to model may have many DoF, this space will be high-dimensional and intractable to model analytically. To address this, we rely on sampled configurations to model the reachable volumes. Our approach consists of four main steps shown in Figure 6.1: First, we compute approximate reachable volumes $\{RV_1, ..., RV_n\}$ of the trajectory between π_i and π_{i+1}. Second, within each reachable volume RV_i we sample configurations of the robot on the contact manifold and evaluate if the configurations make progress towards the target waypoint, or if they get stuck. Third, we use these stuck samples to identify regions of the contact manifold in which the robot is likely to become stuck. Fourth, we estimate $P_{stuck}(RV_i \rightarrow RV_{i+1})$, the probability that the robot becomes stuck moving from one reachable volume to the next. Finally, we compute $P_{exec}(\pi_i \rightarrow \pi_{i+1})$, the probability of successfully reaching the next waypoint, which we use to compute the probability that the entire path is successfully executed $P_{exec}(\Pi)$.

6.4.1 Approximate reachable volumes

Approximate reachable volumes provide a time-discretized model of the $\mathcal{C}-$ *space* reachable volume of the robot's trajectory. We extend the concept of $\mathcal{C}-$ *space* reachable volumes presented in [95], which models reachable configurations of a kinematic linkage given the possible values of joints in the linkage. Instead of a linkage of joints, ours are a linkage of configurations of the robot, which we call a *trajectory-linkage* $l = \{(q_0, t_0), (q_1, t1), ..., (q_n, t_n)\}$ consisting of a sequence of configurations in time. The reachable volume of a trajectory-linkage is the set of all possible configurations at each point in time given the previous configurations.

The advantage of this model of reachable volumes is that they can be defined recursively, as we adapt from [95]:

$$RV_t = RV_{t-1} \oplus RV_{t-1 \to t} \tag{6.1}$$

The reachable volume at time t, RV_t, can be defined as the composition of the reachable volume at $t-1$, RV_{t-1}, and the reachable volume of the link from $t-1$ to t, $RV_{t_{i-1} \to t}$. The initial reachable volume RV_{t_0} is defined by the $\epsilon_{reached}$ threshold for reaching a waypoint in the path, while the link between two reachable volumes is constructed from all possible configurations that the robot can reach in one timestep of the controller given a configuration in the previous reachable volume:

$$RV_{t-1 \to t} = \{q + \Delta q + \omega\} \ \forall q \in RV_{t-1} \tag{6.2}$$

where Δq is displacement produced by the controller as a function of previous configuration q and target waypoint π_i, and ω is the maximum displacement caused by actuator error drawn from $\mathbf{F}(\Delta q)$. Of course, this definition of reachable volumes is not useful on its own; if t, Δq or ω is unbounded, then reachable volumes will grow to encompass the entire $\mathcal{C}-$ *space* of the robot. However, we seek to model the reachable volume of a real robot, one for which Δq is bounded by limits to velocity and timestep Δt, and ω is bounded by limits to actuator error. We show that even if t is not bounded, the robot's controllers ensure that the reachable volume remains bounded, provided that we assume the controllers used are stable and thus converge to the target. We present the calculation of reachable volumes in terms of a robot using a common PD controller for each actuator, however, this calculation can be adapted for other control methods.

First, we show how to calculate the bounds of the linkage from $RV_{t-1 \to t}$. We compute the bounds $B_{d,t}$ for each dimension d of the configuration of the robot, given the bounds of the dimension in the previous reachable volume $B_{d,t-1}$ and target waypoint π_{i+1}. Here, $\pi_{i+1,d}$ is the value of $d-th$ dimension of the target waypoint π_{i+1}, while $\min B_{d,t-1}$ and $\min B_{d,t-1}$ are the minimum and maximum values of the bounds $B_{d,t-1}$, respectively.

First, we compute the error for lower and upper bounds:

$$E_{\min B_d} = \pi_{i+1,d} - \min B_{d,t-1} \tag{6.3}$$
$$E_{\max B_d} = \pi_{i+1,d} - \max B_{d,t-1} \tag{6.4}$$

From this, we compute the nominal velocity command for each bound, incorporating the gains of the $d-th$ dimension actuator's PD-controller, K_p and K_d:

$$\Delta q_{\min B_{d,t-1}} = \textsc{clamp}(K_p E_{\min B_d} + K_d \dot{E}_{\min B_d}, \Delta t) \tag{6.5}$$
$$\Delta q_{\max B_{d,t-1}} = \textsc{clamp}(K_p E_{\max B_d} + K_d \dot{E}_{\max B_d}, \Delta t) \tag{6.6}$$

Here, Clamp represents the applications of actuator limits (i.e., velocity or acceleration limits) over timestep Δt. We combine with noise drawn from $\mathbf{F}(\Delta q)$ to compute the new bounds. *Theorem 1:* The new bounds are given by:

$$\min B_{d,t} = \min B_{d,t-1} + \Delta q_{\min B_{d,t-1}} + \min \mathbf{F}(\Delta q_{\min B_{d,t-1}}) \tag{6.7}$$
$$\max B_{d,t} = \max B_{d,t-1} + \Delta q_{\max B_{d,t-1}} + \max \mathbf{F}(\Delta q_{\max B_{d,t-1}}) \tag{6.8}$$

Proof: Provided that the previous reachable volume(s) $RV_{t-1}, ..., RV_{t_0}$ are convex, and $\mathbf{F}(\Delta q)$ has convex finite support, the resulting reachable volume is convex as it is the Minkowski sum of convex sets. By definition, the initial reachable volume RV_{t_0} defined by $\epsilon_{reached}$ is convex. The link between two sequential reachable volumes is also convex, as it is the Minkowski sum of the affine transformation of the initial reachable volume and noise drawn from $\mathbf{F}$. By induction, all reachable volumes are convex.

It can be shown that the Minkowski sum of convex hulls of convex sets is equal to the convex hull of the Minkowski sum of the sets. For each dimension, the bounds are calculated from this convex hull. Equation 6.7 is derived by summing the bounds of the affine transformation induced by the controller (a convex set) with the bounds of noise induced by $\mathbf{F}$ (also a convex set). □

It is also important to show in an environment without obstacles, the robot is guaranteed to reach within $\epsilon_{reached}$ of π_{i+1} in finite time, which ensures that the number of reachable volumes is finite. This requires that controllers used be stable, and that failure to reach the next waypoint cannot occur as the result of other constraints (ex. torque limits).

Theorem 2: The set of approximate reachable volumes covers the true reachable volume of the robot for the motion between π_i and π_{i+1}.

Proof: It is clear that the affine transformation induced by the controller always makes progress in each timestep towards π_{i+1}, as the controller used is stable. Once the reachable volumes contain the $\epsilon_{reached}$-ball centered at π_{i+1}, we know the reachable volumes will not advance further, since the controller will always move the robot towards the target configuration. Thus, the reachable volumes $RV_{t_0}, ..., RV_{t_n}$ will eventually cover the entire true reachable volume of $\mathcal{C} - space$ for the robot moving from π_i to π_{i+1}. □

6.4.2 Reachable volume sampling

Within each reachable volume RV_i, we need to model the potential distribution of robot configurations. While the effects of uncertainty in the robot's actuators can be modeled to generate such a distribution, this does not incorporate the effects of contact or collision which can make such a distribution lose support in one or more dimensions. Instead, to ensure coverage of the reachable volume, we uniformly sample configurations Q_{RV_i} proportional to the measure (i.e., volume) of RV_i. This sampling is controlled by parameter *density*, which represents the number of samples to draw for a unit of measure. For each configuration $q \in Q_{RV_i}$, we check for collision and assign one of three labels: *colliding* for samples in collision with the environment or self-collision, *near-collision* for samples close to collision, and *free* for the remaining samples. Ideally, we would directly identify near-collision configurations based on proximity to the contact manifold; however, exactly modeling this manifold is difficult. Rather, we approximate $\mathcal{C}-space$ proximity to collision using workspace proximity by checking collision against an expanded version of the environment and robot geometry.

6.4.3 Contact manifold characterization

We wish to characterize which parts of the contact manifold lead to the robot becoming stuck. Sampling within reachable volumes allows us to identify near-collision configurations, but since the contact manifold is zero-volume, sampling will not produce contacting configurations directly. To generate these contacting configurations, we iteratively project near-collision configurations towards contact using the Jacobian $\mathbf{J}$ pseudoinverse of the robot for configuration q:

$$\Delta q = \mathbf{J}(q, p_{closest})^{+}[\Delta p_{closest}^{T}]^{T} \tag{6.9}$$

We identify $p_{closest}$, the point on the robot closest to collision, and compute $\Delta p_{closest}$, the motion that moves $p_{closest}$ towards contact. We iteratively project the configuration towards contact until either contact is achieved or the configuration leaves the current reachable volume RV_i.

Once all contacting configurations $Q_{contacting}$ have been identified, we briefly forward-simulate each configuration $q_{contacting} \in Q_{contacting}$ towards waypoint π_{i+1} for Δt to produce Q_{result}. While we used our approximate kinematic simulator (see Appendix B) to provide simulation, any simulation method capable of modeling contact and robot compliance can be used for this process. We identify which configurations become stuck by checking $dist(q_{contacting}, \pi_{i+1}) - dist(q_{result}, \pi_{i+1}) \geq \Delta_{stuck}$ and assign labels *sliding* for those that made sufficient progress towards π_{i+1} and *stuck* to those that have not. Note that this alone is not enough; a robot may slide in contact for some distance before becoming stuck, and we need to identify these sliding regions that lead to being stuck. Doing so exactly, like many operations involving the contact manifold, is intractable in many cases. Instead, we check if a stuck region is "downstream" of a sliding region. We perform this check by rejecting the nominal motion of the

contacting configurations defined by $\overrightarrow{q_{contacting}\pi_{i+1}}$ onto the nominal motion of the trajectory $\overrightarrow{\pi_i\pi_{i+1}}$. For each $q_{sliding}$ with rejected motion $\overrightarrow{q_{sliding}\pi_{i+1}}_{rejected}$, we compare against each q_{stuck} with rejected motion $\overrightarrow{q_{stuck}\pi_{i+1}}_{rejected}$. If there exists a q_{stuck} such that $|\overrightarrow{q_{stuck}\pi_{i+1}}_{rejected}| < |\overrightarrow{q_{sliding}\pi_{i+1}}_{rejected}|$ and the angle between $\overrightarrow{q_{stuck}\pi_{i+1}}_{rejected}$ and $\overrightarrow{q_{sliding}\pi_{i+1}}_{rejected}$ is less than small angle θ, then we identify $q_{sliding}$ as *becoming stuck*. Intuitively, this checks if sliding configurations are likely to pass through regions in which they will become stuck.

6.4.4 Stuck probability estimation

To estimate the probability that the robot becomes stuck for each pair of sequential reachable volumes $P_{stuck}(RV_i \rightarrow RV_{i+1})$, we estimate the probability that Q_{RV_i}, the sampled configurations in RV_i, reach stuck regions in RV_{i+1} in the next timestep. To do so, we compute bounds on the *image* of each $q \in Q_{RV_i}, q \notin Q_{stuck}$, i.e., the bounds on the distribution of possible future configurations. As our mode of uncertainty is additive, we can model the image as a distribution centered on q_{next}, the configuration resulting from Δt of noise-free motion, with bounds corresponding to the maximum noisy displacement in each dimension. To ensure that this check is robust to sampling density, we ensure that the bounds are no smaller than the expected distance between samples $d_{expected} = 1/density^{1/dimensions}$. Within the image, we check for sliding and stuck regions identified in RV_{i+1}, and record the relevant q_{stuck} and $q_{sliding}$ as $Q_{RV_{i+1},relevant}$. If no stuck configurations are found, we assume that the region of RV_i represented by q does not become stuck. For q with corresponding stuck configurations, we integrate the image of q over the stuck regions of RV_{i+1}.

Exactly integrating a multi-dimensional probability distribution in the presence of obstacles is challenging, so we perform numerical integration using a Monte-Carlo approach. For a given number of iterations, we draw a possible noisy future configuration q_{future} and check if it falls within a stuck region of RV_{i+1}. Note that these stuck regions are regions of the contact manifold, and thus have zero volume, so no q_{future} will fall within one. Instead, we must check if the motion of the robot from q to q_{future} makes contact in one of these regions. Since we wish to avoid the cost of simulating motion from q to q_{future} to check for contact, we must approximate this check. We know that any q_{future} in collision means that the robot must reach the contact manifold, and we retrieve the closest configuration $q_{close} \in Q_{RV_{i+1},relevant}$ as a heuristic to determine if q_{future} belongs to a stuck region. If q_{close} is stuck, we determine that q_{future} is stuck. The probability of a configuration q becoming stuck is:

$$P_{stuck}(q \rightarrow RV_{i+1}) = |\{q_{future,stuck}\}|/|\{q_{future}\}| \tag{6.10}$$

We sum these probabilities for all collision-free $q \in RV_i$ to compute probability that the robot becomes stuck trying to reach the next reachable volume:

$$P_{stuck}(RV_i \to RV_{i+1}) = \sum_{q \in RV_i, q \in \mathcal{C}-free} (P_{stuck}(q \to RV_{i+1})) \tag{6.11}$$

Performing this estimate for each pair of sequential reachable volumes, we compute the probability that the robot reaches the next waypoint. Reaching the next waypoint is equivalent to the probability of *not* becoming stuck for all pairs of squential reachable volumes:

$$P_{exec}(\pi_i \to \pi_{i+1}) = \Pi_{i=1}^{n-1}(1 - P_{stuck}(RV_i \to RV_{i+1})) \tag{6.12}$$

Finally, we can compute the probability that the entire path can be executed:

$$P_{exec}(\Pi) = \Pi_{i=1}^{goal-1}(P_{exec}(\pi_i \to \pi_{i+1})) \tag{6.13}$$

6.5 Results

We present results of testing our path quality metric in $SE(2)$ and $SE(3)$ environments. Since our metric operates over pairs of adjacent waypoints independently, our experiments are designed to reflect the most challenging pairs of adjacent waypoints from the motion planning tasks of Section 5.4. To provide a ground-truth metric for path quality, we execute the provided paths using a Kuka LBR IIWA 7 robot. Since this robot is not inherently passively compliant, paths were executed using the onboard cartesian impedance controller to provide active compliance. As the kinematic simulator used to check if motions become stuck does not consider friction, we use *contact motion controllers* to reduce contact forces (see Appendix C for details on the controllers). We present statistical results over a range of actuation uncertainty and values of $\epsilon_{reached}$ and show that our metric can provide reasonable assessments of path quality compared to real robot performance of the task. We discuss the efficiency and accuracy of our proposed metric against several simpler approaches. All metric evaluation was performed using a 3.4 GHz 16-thread R7-1700X processor.

For both $SE(3)$ and $SE(2)$ tasks, we used the same range of $\epsilon_{reached} = \{0.125, 0.1875, 0.25\}$ and actuator error $\alpha = \{0.0625, 0.125, 0.25\}$. To accommodate for the small translation component of the tasks (on the order of tens of centimeters at most) we weighted translation by a factor of 10 (i.e., $\epsilon_{reached} = 0.25$ is equivalent to 0.25 radians or 2.5 centimeters). Actuator error is modelled in the form of velocity noise, with linear velocity similarly weighted to $1/10$, while angular velocity is weighted by $1/4$ as in Section 5.4. For a given value of α, velocity noise for actuator d is drawn from a zero-mean truncated normal distribution with bounds $[-lim_d, lim_d]$ and standard deviation σ_d proportional to the desired actuator velocity $\dot{q}_d$ and maximum actuator velocity $\max\dot{q}_d$:

$$lim_d = \max(\alpha \dot{q}_d, \alpha \frac{1}{4} \max \dot{q}_d) \tag{6.14}$$

$$\sigma_d = \frac{1}{2} lim_d \tag{6.15}$$

While this model, a further development of those used in Section 5.4, differs from the zero-mean normal distributions commonly used in the literature, we believe a bounded distribution better models the reality of robot actuators; it is also required by our model of reachable volumes, as a pure normal distribution does not have bounded support and thus reachable volumes would cover the entire $\mathcal{C} - space$. Additionally, we believe the combination of noise proportional to commanded actuator velocity and a fixed noise floor better approximates real actuator behavior than either of the previous models we have explored.

The ground truth likelihood of successfully completing the task was determined by performing 100 attempts of each combination of $\epsilon_{reached}$ and α. We compare this "true" likelihood against P_{exec} estimated by our method "Metric" and several simpler simulation-based approaches that rely on our approximate kinematic simulator to evaluate robot motion:

1. Simulation (scaled) – Simulate the motion of the robot from a randomly sampled configuration in RV_{t_0} of the current waypoint until it either reaches within $\epsilon_{reached}$ of the next waypoint or fails to make progress and becomes stuck. This simulation is repeated proportional to the measure of the largest reachable volume encountered on the provided path. The estimated P_{exec} is the percentage of iterations which reach the target waypoint. This method provides a means to conservatively estimate the number of simulation iterations needed.

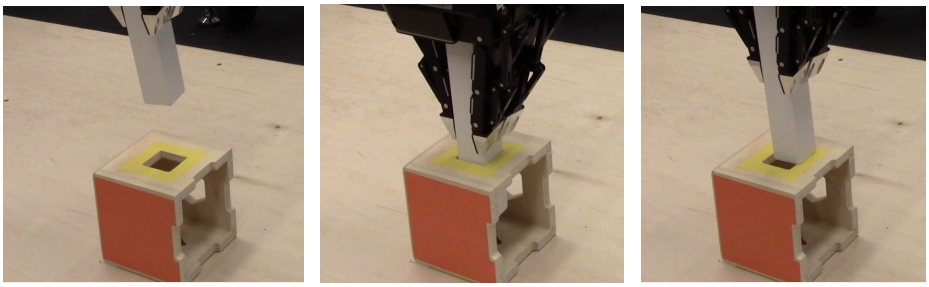
2. Sampling-driven – Compute the reachable volumes from current to target waypoint, and sample within them, checking for stuck samples in each reachable volume. Record the last reachable volume that contains a stuck sample as RV_s. If a stuck sample is detected, simulate all free samples in RV_{t_0} until they: 1) reach RV_{s+1}, 2) within $\epsilon_{reached}$ of the target waypoint, or 3) fail to make progress and become stuck. This method offers a principled way to detect where the robot may become stuck and simulation may be necessary, while offering a set of starting configurations.

For purposes of comparison, we also include Simulation (baseline), in which we assess the true likelihood of successfully completing the path in our simulator by simulating with a high number of simulations. This baseline is important, since it shows any differences between the behavior of the simulated and real robots.

It is important to note that the behavior modeled by the kinematic simulator will not exactly correspond to the behavior of the real robot. The previously-developed compliant motion controllers mitigate the effects of friction during sliding contact, but for the controller to adapt the motion of the robot, it must

get stuck *first*. Therefore, to detect if the robot is actually stuck, rather than merely in the process of being un-stuck by the controller, we must check if the robot fails to make sufficient forward progress over several successive iterations. While a similar check can be incorporated into the kinematic simulator, these are not directly equivalent since the kinematic simulator will never get stuck as the result of friction. Requiring multiple successive stuck iterations (in our experience, 2 successive iterations is reasonable) may cause the simulator to overestimate the likelihood of successfully executing the path, since given time, the effects of actuator error may "un-stick" the simulated robot.

6.5.1 $SE(3)$ Peg-in-hole



(a) (b) (c)

Figure 6.2: The $SE(3)$ peg-in-hole task involves moving from the start (a) to the bottom of the hole. (b) An example of a successful execution of the task, in which the peg enters the hole. (c) An example of a failed execution, in which the peg becomes stuck and fails to reach the hole.

In $SE(3)$ peg-in-hole, a version of the classical peg-in-hole task [42] shown in Figure 6.2, the free-flying 6-DoF robot "peg" must reach the bottom of the hole. This task is difficult for robots with actuation uncertainty, as the hole is less than 10% wider than the peg. For this experiment, Simulation (baseline) was performed with 10000 simulations from the initial to target waypoint, and sampling density parameter *density* was set so that at the lowest values of $\epsilon_{reached}$ and α the largest reachable volume contained 500 samples; thus Simulation (scaled) and Sampling-driven performed 500 simulations. For this task, at most 21 reachable volumes are computed, depending on $\epsilon_{reached}$ and α, and thus Sampling-driven and Metric must sample and assess approximately 10000 configurations. Note that this sampling density is not particularly high, however, higher sampling densities would result in excessive computation time.

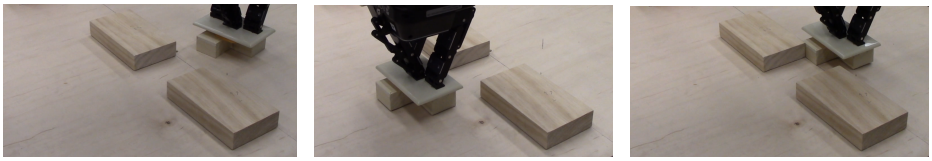
As shown in Table 6.1, Simulation (scaled) and Sampling-driven produce similar assessments of P_{exec}; this is to be expected since these two methods perform identical amounts of simulation. Both estimate similar P_{exec} to the simulation baseline, despite performing $1/20$ of the simulation. However, true P_{exec} estimated from real robot experiments often differs significantly from the

Method	$\epsilon_{reached} = 0.125$			$\epsilon_{reached} = 0.1875$			$\epsilon_{reached} = 0.25$		
	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25
Ground truth	0.91 [0.0]	0.97 [0.0]	0.92 [0.0]	0.82 [0.0]	0.91 [0.0]	0.84 [0.0]	0.85 [0.0]	0.79 [0.0]	0.84 [0.0]
Simulation (baseline)	1.0 [0.00]	0.99 [0.00]	0.80 [0.00]	0.99 [0.00]	0.99 [0.00]	0.86 [0.00]	0.99 [0.00]	0.98 [0.00]	0.88 [0.00]
Simulation (scaled)	1.0 [0.00]	0.99 [0.00]	0.80 [0.02]	1.0 [0.00]	0.99 [0.00]	0.86 [0.01]	0.99 [0.00]	0.98 [0.01]	0.88 [0.02]
Sampling-driven	1.0 [0.00]	0.99 [0.00]	0.84 [0.02]	1.0 [0.00]	0.99 [0.00]	0.89 [0.02]	0.99 [0.0]	0.96 [0.18]	0.88 [0.01]
Proposed Metric	1.0 [0.00]	1.0 [0.0]	0.99 [0.01]	0.99 [0.01]	0.96 [0.02]	0.87 [0.02]	0.88 [0.02]	0.79 [0.04]	0.71 [0.03]

Table 6.1: $SE(3)$ peg-in-hole task success comparison (mean [std.dev.]) between real-world execution of the task. Ground truth percentages are the result of 100 executions of each combination of state-reached threshold $\epsilon_{reached}$ and actuator error α. Assessed execution probability from our method (Proposed Metric) and alternative approaches are averaged over 30 runs.

predicted value. Our proposed Metric produces a closer estimate overall to the true P_{exec}, but this estimate is not uniformly conservative as we would desire. We discuss these results, and compare the efficiency of these methods in detail in Section 6.6.

6.5.2 $SE(2)$ Through-passage



(a) (b) (c)

Figure 6.3: The $SE(2)$ through-passage task involves moving from the start (a) through the passage. (b) An example of a successful execution of the task, in which the L-block clears the passage. (c) An example of a failed execution, in which the block becomes stuck.

In $SE(2)$ through-passage, shown in Figure 6.3, the free-flying 3-DoF robot "L-block" must pass hole. This task is difficult for robots with actuation uncertainty, as the hole is only a few millimeters wider than the peg. Note that due to the asymmetry of the block, rotations to one side or the other result in significantly different likelihoods of success. For this experiment, Simulation (baseline) was performed with 2000 simulations from the initial to target way-

Method	$\epsilon_{reached} = 0.125$			$\epsilon_{reached} = 0.1875$			$\epsilon_{reached} = 0.25$		
	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25
Ground truth	0.74 [0.0]	0.74 [0.0]	0.8 [0.0]	0.68 [0.0]	0.63 [0.0]	0.71 [0.0]	0.61 [0.0]	0.45 [0.0]	0.54 [0.0]
Simulation (baseline)	1.0 [0.00]	0.96 [0.01]	0.63 [0.02]	1.0 [0.00]	0.99 [0.00]	0.72 [0.01]	1.0 [0.00]	0.99 [0.0]	0.77 [0.02]
Simulation (scaled)	1.0 [0.00]	0.95 [0.01]	0.63 [0.03]	1.0 [0.00]	0.99 [0.01]	0.73 [0.03]	1.0 [0.00]	0.99 [0.01]	0.77 [0.03]
Sampling-driven	1.0 [0.00]	1.0 [0.00]	0.79 [0.04]	1.0 [0.00]	0.99 [0.01]	0.80 [0.02]	1.0 [0.00]	0.99 [0.00]	0.82 [0.02]
Proposed Metric	0.98 [0.03]	0.79 [0.10]	0.39 [0.06]	1.0 [0.00]	0.87 [0.10]	0.38 [0.07]	1.0 [0.01]	0.78 [0.10]	0.35 [0.05]

Table 6.2: $SE(2)$ through-hole task success comparison (mean [std.dev.]) between real-world execution of the task. Ground truth percentages are the result of 100 executions of each combination of state-reached threshold $\epsilon_{reached}$ and actuator error α. Assessed execution probability from our method (Proposed Metric) and alternative approaches are averaged over 30 runs.

point, and sampling density parameter *density* was set so that at the lowest values of $\epsilon_{reached}$ and α the largest reachable volume contained 250 samples; thus Simulation (scaled) and Sampling-driven performed 250 simulations. For this task, at most 31 reachable volumes are computed, depending on $\epsilon_{reached}$ and α, and thus Sampling-driven and Metric must sample and assess approximately 7800 configurations. Like the $SE(3)$ case, this sampling density is low; higher sampling densities would result in excessive computation time.

As shown in Table 6.2, Simulation (scaled) produces nearly identical assessments of P_{exec} to baseline simulation, despite performing a fraction of the simulation. Unexpectedly, Sampling-driven produces different estimates at high uncertainty. Similarly, Metric consistently produces lower estimates of P_{exec} in mid- and high-uncertainty cases. These differences suggest that the chosen sampling density is too low. Choosing a higher density would result in significantly higher computation times, however. When compared to the true P_{exec}, all methods show significant inaccuracy. In all cases, the true P_{exec} is lower than predicted at low- and mid-uncertainty. Only at high uncertainty does one method, Metric, produce more conservative estimates. We discuss the efficiency of the presented methods and reasons for their inaccuracy in Section 6.6.

6.6 Discussion

While our experiments suggest that the proposed methods show promise, they do not produce uniformly positive conclusions. Both our proposed method and several baseline methods struggle to accurately predict the true likelihood that our example tasks will be successfully executed. In light of our experience

planning for such tasks, we believe the most important issue is the mismatch between the behavior of the idealized simulator and controllers and the behavior of the real robot with its own distinct control scheme. This is shown in the contrast between the relative similarity of baseline simulation and metric-assessed P_{exec}, while the ground truth value may differ significantly. In our previous experiments, we sought to develop robot controllers that would allow reliable execution of planned motions. Here, however, producing accurate assessments of P_{exec} requires that the behavior of simulated and real robots be nearly identical: there cannot be systematic cases where one succeeds and the other fails. The computational efficiency of our approximate kinematic simulator makes it possible to cheaply model compliant robot behavior, but on its own, it cannot accurately model tasks that involve friction or robot dynamics.

Another fundamental limitation of the presented method is the reliance on uniform sampling. Uniform sampling is clearly suboptimal; we know from experience and in simulated and real environments that potential robot configurations are concentrated near the medial axis of the path. However, accurately modeling a more accurate distribution in the presence of obstacles is difficult. Uniform sampling, while inferior to using the true distribution, ensures that we do not introduce unexpected artifacts in our sampling. A consequence of using uniform sampling is that the metric may underestimate P_{exec} in case where samples near the bounds of reachable volumes become stuck, as uniform sampling will overestimate the likelihood of the robot being in these regions. We believe this effect is particularly visible in the highest uncertainty cases of our $SE(2)$ experiment as seen in Table 6.2 where our proposed metric significantly underestimates the true likelihood of reaching the target waypoint.

6.6.1 Efficiency

To be practically useful as a method for assessing the quality of a path in a motion planner or trajectory optimizer, a path quality metric must be efficient. In our tests, we have explicitly limited sampling density and simulation iterations in an effort to limit computation time to reasonable durations. Higher accuracy from sampling-based methods is of no use if the computation time required by greater density exceeds that of pure simulation. The computation time required by all methods are shown in Table 6.3 and Table 6.4 for $SE(3)$ and $SE(2)$ tasks, respectively. All methods exhibit a significant degree of parallelism: simulations and sampling/assessment may be performed in parallel, as inter-dependencies between simulation iterations and samples are minimal. To the greatest extent practical, the methods presented have been parallelized through the use of OpenMP directives [96]. Further increases to performance would require significant algorithmic changes.

For $SE(3)$, Simulation (scaled) is clearly the most efficient method, requiring approximately a quarter of the computation time of the next-most efficient method. However, as can be seen in both $SE(3)$ and $SE(2)$ tasks, all methods using simulation require significantly more computation time as $\epsilon_{reached}$ and α increase. This results from the increased contact present at higher parameter

values. Our kinematic simulator is designed to ensure that free-space motion is evaluated as cheaply as possible (in practice, requiring little more than intermediate collision checks), while motion in contact requires collision resolution to be performed at every simulation iteration.

Method	$\epsilon_{reached} = 0.125$			$\epsilon_{reached} = 0.1875$			$\epsilon_{reached} = 0.25$		
	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25
Simulation (baseline)	4.99 [0.05]	6.26 [0.08]	13.08 [0.19]	4.78 [0.07]	6.33 [0.08]	12.62 [0.19]	5.43 [0.09]	6.99 [0.10]	12.63 [0.17]
Simulation (scaled)	0.26 [0.01]	0.32 [0.01]	0.66 [0.03]	0.24 [0.01]	0.34 [0.02]	0.65 [0.03]	0.29 [0.02]	0.35 [0.02]	0.64 [0.03]
Sampling-driven	1.48 [0.08]	1.47 [0.15]	2.08 [0.08]	2.0 [0.10]	2.03 [0.08]	2.38 [0.08]	1.82 [0.06]	1.93 [0.09]	2.23 [0.06]
Proposed Metric	1.45 [0.06]	1.44 [0.10]	1.93 [0.08]	1.96 [0.11]	2.08 [0.10]	2.25 [0.07]	1.99 [0.07]	2.16 [0.11]	2.32 [0.06]

Table 6.3: $SE(3)$ peg-in-hole task computation time (seconds) comparison (mean [std.dev.]) between our method (Proposed Metric) and alternative approaches averaged over 30 runs of each combination of state-reached threshold $\epsilon_{reached}$ and actuator error α.

Method	$\epsilon_{reached} = 0.125$			$\epsilon_{reached} = 0.1875$			$\epsilon_{reached} = 0.25$		
	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25	$\alpha =$ 0.0625	$\alpha =$ 0.125	$\alpha =$ 0.25
Simulation (baseline)	7.32 [0.19]	12.60 [0.22]	19.89 [0.48]	7.45 [0.17]	12.47 [0.31]	20.31 [0.54]	7.53 [0.16]	12.56 [0.25]	20.27 [0.43]
Simulation (scaled)	1.79 [0.06]	3.06 [0.11]	4.91 [0.16]	1.84 [0.08]	3.06 [0.09]	5.00 [0.18]	1.84 [0.07]	3.13 [0.10]	5.02 [0.15]
Sampling-driven	2.47 [0.24]	3.82 [0.30]	7.36 [0.47]	2.46 [0.32]	3.91 [0.47]	7.58 [0.30]	2.39 [0.16]	3.85 [0.57]	7.50 [0.43]
Proposed Metric	2.42 [0.08]	3.00 [0.15]	3.53 [0.13]	2.45 [0.06]	2.85 [0.11]	3.48 [0.11]	2.36 [0.07]	2.74 [0.10]	3.38 [0.10]

Table 6.4: $SE(2)$ through-hole task computation time (seconds) comparison (mean [std.dev.]) between our method (Proposed Metric) and alternative approaches averaged over 30 runs of each combination of state-reached threshold $\epsilon_{reached}$ and actuator error α.

Unlike the simulation-based methods, our proposed method increases in computation time at a slower rate, as a combination of the increasing volume of the computed reachable volumes and the increasing number of *near-contact* sampled configurations which are projected to contact and evaluated for stuck/sliding. This can be see in the $SE(2)$ task, where simulation-based methods that start with the lowest computation time quickly rise above the time required by the Metric approach. This shows that our sampling-based method provides an ef-

ficient alternative in case where simulation-based approaches become too expensive. In particular, this approach shows particular promise efficiency-wise if used with a more expensive simulator. By avoiding prolonged simulation, the sampling-based approach may show significant efficiency improvements in comparison to paths evaluated in a dynamic simulator such as Gazebo [94].

6.6.2 Future improvements

There are several avenues for further development of the proposed methods. A limitation of the proposed approach is that reachable volumes may contain volumes of $\mathcal{C} - free$ that are not reachable from previous reachable volumes due to obstacles in the environment. Likewise, since the approximate reachable volumes are computed assuming no obstacles are present, they are limited to modeling visible paths. We briefly discuss two options are available for addressing this limitation.

Using the spatial particle clustering heuristics introduced in Section 5.3.3, we can check that all n samples in a given reachable volume RV_t are close to at least one of the m samples in the previous reachable volume RV_{t-1}. Intuitively, these clustering heuristics reflect the reachability of regions of the environment, so using them to filter which samples are reachable is a natural extension. Indeed, we have explored using this technique but not included it in our results; it requires significant computation time. A naïve implementation requires $\mathrm{O}(mn)$ comparisons, while an optimized implementation using K-d trees for nearest-neighbor search may reduce this complexity to $\mathrm{O}(\log(m)n)$ comparisons. Implementing high-performance K-d trees for configuration spaces containing $SO(3)$, however, is difficult [97].

A more exact representation of the reachable volume of a path can be derived by modeling the actual motion of potential configurations of the robot. Not only does this approach avoid the issue of extraneous free reachable volume, it also allows non-visible paths to be evaluated. A potential avenue for such an approach is to adopt ideas from the Unscented Kalman Filter (UKF) like [75], and use the propagation of possible configurations as *sigma points* for the UKF. Other belief-space approaches use Extended Kalman Filters to propagate belief [98]. This would provide both an accurate model of the true reachable volume and a useful non-uniform distribution within the volume that might better represent the likely configurations of the robot. Unfortunately, it is unclear how to implement such an approach without heavy reliance on computationally-expensive simulation. In such a case, the amount of simulation alone could be equivalent to that needed to directly estimate the likelihood of successfully completing the path.

6.7 Conclusion

We have developed a pair of methods for assessing the robustness of paths executed by robots with actuation uncertainty. Both methods incorporate en-

vironment contact and compliance of the robot to distinguish between contact that allows the robot to *slide* and complete the path, and contact that results in the robot becoming *stuck*. We model the possible reachable volume of $\mathcal{C}-space$ covered while executing a provided path using a set of *approximate reachable volumes*. Within each reachable volume, we use sampling to identify regions of the contact manifold on which the robot may become stuck. Using these regions, we estimate the likelihood that the robot will successfully execute the provided path. We have applied our methods to several real world robot tasks in $SE(3)$ and $SE(2)$ and compare against simpler simulation-based methods. We show that our proposed methods, while not uniformly superior to pure simulation, offer a promising avenue for efficient assessment of path robustness. In addition, we discuss limitations of the proposed work and directions for future improvements.

Acknowledgements

Thanks to Nathan Hughes, who formulated the model of reachable volumes and prototyped parts of the proposed metric.

Chapter 7

Discussion and Future Work

This thesis makes contributions to the areas of motion planning with deformable objects and environments and manipulation with uncertainty, but these tasks remain challenging for robots. We have focused on tasks where issues of sensing, modeling, planning, and control are often inseparable: even the best planned motion may be for naught if the controllers attempting to perform it are poor. As a result, in our search for methods that allow robots to perform useful practical tasks, addressing the tasks we wish to perform often involves the development of a combination of new models, planning approaches, and execution methods.

We have developed approximate models and cost functions that allow practical and efficient motion planning in elastic deformable environments to minimize deformation. These environments are representative of those encountered in important real-world tasks such as robotic assembly and surgery. Previous work in this area was limited to local optimization or relied upon extensive precomputation. Beyond simply modeling physical properties, we have incorporated qualitative information about deformable objects into our models and provided means for expert users to accurately build these models from their expert knowledge of the task. Combining quantitative properties and qualitative characteristics is a key step in making robotic manipulation of deformable objects and environments practical for the kinds of complex heterogeneous environments encountered in industrial and medical tasks.

Despite the success of our approximate model, we need not forsake more principled and accurate simulation methods. The voxel-based model presented in Chapter 3 is a coarse approximation of real deformable objects. For objects that are not completely elastic or are not fixed in place, better models are needed. Higher fidelity models, such as FEM simulation, can be used to improve lightweight discretized models. At the most basic level, more accurate simulation can be used to compute parameters for our models that capture higher-order characteristics like fragility. Expensive high-fidelity simulation can

be used up front to train models such as GMMs [2] or neural networks which then predict the severity of object deformation or the behavior of the deformable objects. For example, this could offer a means of incorporating object flexibility and motion into our lightweight models, where a trained model predicts the amount of bending or movement that will result for a given deformation. This could be combined with our cost function to produce conservative estimates of cost that account for the possibility of object bending or movement. A similar opportunity exists with puncture detection: if the likelihood of a path resulting in puncture could be predicted using a trained model, it would greatly reduce the runtime expense of planning using our topology-based puncture check.

Some of the limitations imposed by our models may be addressed by closing the loop of sensing, modeling, planning, and execution. Rapid adaptation of planned paths and replanning as-needed offer promise to reliably perform tasks in dynamic deformable environments without paying the up-front high costs of higher-fidelity models. Indeed, work in visual servoing and control has produced FEM models that may be evaluated in realtime; while these are not fast enough to use in a planner, they are fast enough to use inside a controller that adapts the planned path to real object behavior encountered during execution. Parallel work on the development of efficient approximate models for control of deformable objects [25, 99] may offer an alternate approach. Instead of planning motions in a purely kinematic sense, planning incorporating the behavior of controllers and producing an *execution policy* rather than a *path* offers a potential way to combine execution, sensing, and as-needed replanning.

Policy-based planning and execution form the core inspiration of our work planning manipulation for robots with actuation uncertainty. Even the most reliable industrial robot will diverge from the exact path produced by a planner; let alone a robot with actuator error. Intuitively, we want a planner to generate behavior for a range of possible robot states, and for that behavior to adapt to reflect the success or failure of execution. The planner presented in Chapter 5 provides an efficient means for generating policies that adapt online to reflect the true behavior encountered during execution. Unlike previous work in this area, we have developed a planner that can safely use and exploit contact and compliance. Just like working with deformable objects, we have turned the problem of contact into a strength. Our robots can automatically leverage contact like that proposed in seminal planning work [42] without specifying which, if any, contact to use.

Like our voxel-based model that makes planning for deformable objects efficient, planning with contact is fundamentally enabled by our development of a fast approximate kinematic simulator (see Appendix B). Existing kinodynamic planning using simulated high-DoF robots rarely incorporates contact, True dynamic simulation in contact is often computationally expensive. Without this simulator, it would be impractical to plan these motions; rather than the seconds it takes for our planner to produce its first solutions, using a full dynamics simulator would take minutes or hours. This approximate simulator is not without its limitations, many of which can (and often must) be addressed to produce reliable robot behavior.

Of course, the planner introduced in Chapter 5 is not perfect. It strikes one possible balance in a set of options: how much to model and simulate up front; exploration of multiple alternative motions; and reliance on post-processing and smart execution. Clearly, different combinations of robots and tasks could benefit from different choices: our experience shows that many tasks can be planned without explicitly modelling uncertainty at all, provided a sufficient diversity of possible motions is encoded in the policy. Selecting the right balance for a given task is an important area for future investigation. The role of different controllers and control modes is also one to explore. Reasoning over different controllers together with uncertainty offers the ability to selectively change the certainty of the robot's motion, which could greatly improve the ability of robots to perform precise manipulation without sacrificing speed. Human motion often combines fast, imprecise motion with slow, fine motion. Can we use techniques from planning with uncertainty to produce safe human-like motion on common robots?

The limitations of our planning work inspire us to explore the problem from another direction. In Chapter 6 we address the complement of the planning problem. Instead of planning manipulation in the face of uncertainty, we assess the resilience of a desired motion to the addition of uncertainty. This problem offers a deceptively simple naïve solution: simply execute the motion for a sufficient number of iterations. Of course, executing every planned path with a robot is impractical, perhaps even dangerous, and defeats the purpose of generating robust motion. Simulation alone is also no ideal solution. Accurate simulation is expensive, and ensuring that simulation accurately captures the robustness of *all* parts of a desired motion is difficult.

This challenge drives us to explore the characteristics of the space used by the robot to perform the motion. Reachable volumes provide a discretization in time, while sampling allows us to consider continuous regions of the configuration space in terms of representative samples. Using sampling, we can characterize the relevant regions of the contact manifold, and estimate the probability that a robot will encounter troublesome regions. Though this work provides only the first steps towards a robust metric of path quality, it shows promise for a method that does not rely on extensive simulation.

Our work towards developing an efficient metric for path robustness presented in Chapter 6 is illustrative of the many challenges posed trying to model and perform complex tasks on practical robots. High-DoF robots render search-based planning and fixed discretization impractical, so we rely on sampling. Yet, this sampling adds new problems. Fewer samples reduce computation time to practical levels, yet often introduce artifacts and poor coverage of the space. Often we want not just a specific sample, but a representation of a region of the configuration space. Can a new approach, perhaps relying on advances in topology and geometry, change how we model high-dimensional space?

A troubling consequence of the integration of models, planning, and execution is a lack of theoretical guarantees. This is an issue even when building from planners and controllers with well-known properties. The topology-based puncture check for deformable objects introduced in Chapter 4 becomes a con-

straint on the entire path. While the theoretical consequences of this check have not been explored in depth, discussions suggest that it compromises any completeness or optimality guarantees of a planner using it. Likewise, our methods for planning manipulation with uncertainty in Chapter 5 combine techniques from planning, execution, and state estimation with useful theoretical properties. Yet, their combination creates an entirely new space, in which we must reason over their combined behavior. [58] is a rare example of a method in this area with completeness guarantees, although the success of trajectory optimizers suggests that theoretical properties may be unnecessary for practical utility. Can we prove completeness or optimality for these planning approaches? These are important topics to address in the future.

Bibliography

[1] Maris, B., Botturi, D., Fiorini, P.: Trajectory planning with task constraints in densely filled environments. In: IROS. (October 2010)

[2] Frank, B., Stachniss, C., Abdo, N., Burgard, W.: Efficient motion planning for manipulation robots in environments with deformable objects. In: IROS. (September 2011)

[3] Faure, F., Gilles, B., Bousquet, G., Pai, D.K.: Sparse meshless models of complex deformable solids. ACM Transactions on Graphics (2011) 73–73

[4] Kalman, R.: When is a linear control system optimal? Journal of Basic Engineering (1964)

[5] Ng, A.Y., Russell, S.J.: Algorithms for inverse reinforcement learning. In: ICML. (2000)

[6] Ratliff, N., Silver, D., Bagnell, J.A.D.: Learning to search: functional gradient techniques for imitation learning. Autonomous Robots (2009)

[7] Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: ICML. (2004)

[8] Kalakrishnan, M., Pastor, P., Righetti, L., Schaal, S.: Learning Objective Functions for Manipulation. In: ICRA. (2013)

[9] Karaman, S., Frazzoli, E.: Sampling-based Algorithms for Optimal Motion Planning. The International Journal of Robotics Research **30**(7) (2010) 20

[10] Melchior, N.A., Simmons, R.: Particle rrt for path planning with uncertainty. In: ICRA. (April 2007)

[11] Gibson, S.F.F., Mirtich, B.: A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories (1997)

[12] Alterovitz, R., Goldberg, K.: Motion Planning in Medicine: Optimization and Simulation Algorithms for Image-Guided Procedures (Springer Tracts in Advanced Robotics). Springer (2008)

[13] Couvignou, P., Papanikolopoulos, N., Khosla, P.: Model-based robotic visual servoing. In: American Control Conference (ACC). Volume 1. (1995)

[14] Couvignou, P.A., Papanikolopoulos, N.P., Sullivan, M., Khosla, P.K.: The use of active deformable models in model-based robotic visual servoing. Journal of Intelligent Robotic Systems **17**(2) (1996) 195–221

[15] Lang, J., Pai, D.K., Woodham, R.J.: Acquisition of elastic models for interactive simulation. IJRR **21**(8) (2002) 713–733

[16] Cretu, A., Payeur, P., Petriu, E.: Neural network mapping and clustering of elastic behavior from tactile and range imaging for virtualized reality applications. IEEE TIM **57**(9) (2008) 1918–1928

[17] Barbagli, F., Salisbury, K., Prattichizzo, D.: Dynamic local models for stable multi-contact haptic interaction with deformable objects. In: 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. (2003)

[18] Arikatla, V.S., De, S.: A two-grid iterative approach for real time haptics mediated interactive simulation of deformable objects. In: IEEE Haptics Symposium. (March 2010)

[19] Huang, S., Pan, J., Mulcaire, G., Abbeel, P.: Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In: IROS. (September 2015)

[20] Schulman, J., Ho, J., Lee, C., Abbeel, P.: Learning from demonstrations through the use of non-rigid registration. Springer Tracts in Advanced Robotics **114** (2016) 339–354

[21] Navarro-Alarcon, D., Romero, J.: Visually servoed deformation control by robot manipulators. In: ICRA. (September 2013)

[22] Müller, M., Dorsey, J., McMillan, L., Jagnow, R., Cutler, B.: Stable real-time deformations. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '02, New York, New York, USA (2002)

[23] Irving, G., Teran, J., Fedkiw, R.: Invertible finite elements for robust simulation of large deformation. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '04, New York, New York, USA (2004)

[24] Saha, M., Isto, P.: Motion planning for robotic manipulation of deformable linear objects. In: ICRA. (2006)

[25] Berenson, D.: Manipulation of Deformable Objects Without Modeling and Simulating Deformation. In: IROS. (November 2013)

[26] Fugl, A.R., Jordt, A., Petersen, H.G., Willatzen, M., Koch, R.: 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. (August 2012)

[27] Boonvisut, P., avuolu, M.C.: Estimation of soft tissue mechanical parameters from robotic manipulation data. IEEE/ASME Transactions on Mechatronics **18**(5) (Oct 2013) 1602–1611

[28] Schulman, J., Lee, A., Ho, J., Abbeel, P.: Tracking deformable objects with point clouds. In: ICRA. (May 2013)

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

[30] Frank, B., Stachniss, C., Schmedding, R., Teschner, M., Burgard, W.: Learning object deformation models for robot motion planning. Robotics and Autonomous Systems **62**(8) (2014) 1153–1174

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

[32] Lavalle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University (1998)

[33] Anshelevich, E., Owens, S., Lamiraux, F., Kavraki, L.: Deformable volumes in path planning applications. In: ICRA. (2000)

[34] Burchan Bayazit, O., Amato, N.: Probabilistic roadmap motion planning for deformable objects. In: ICRA. (2002)

[35] Gayle, R., Lin, M., Manocha, D.: Constraint-Based Motion Planning of Deformable Robots. In: ICRA. (2005)

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

[37] Rodriguez, S., Amato, N.: Planning motion in completely deformable environments. In: ICRA. (2006)

[38] Boularias, A., Kober, J., Peters, J.R.: Relative entropy inverse reinforcement learning. In: International Conference on Artificial Intelligence and Statistics. (April 2011)

[39] Park, T., Levine, S.: Inverse optimal control for humanoid locomotion. In: RSS Workshop on Inverse Optimal Control and Robotic Learning from Demonstration. (June 2013)

[40] Ziebart, B.D., Maas, A., Bagnell, J., Dey, A.K.: Maximum Entropy Inverse Reinforcement Learning. In: AAAI. (2008) 1433–1438

[41] Aghasadeghi, N., Bretl, T.: Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In: IROS. (September 2011)

[42] Lozano-Prez, T., Mason, M.T., Taylor, R.H.: Automatic synthesis of fine-motion strategies for robots. IJRR **3**(1) (1984) 3–24

[43] Canny, J.: On computability of fine motion plans. In: ICRA. (May 1989)

[44] Natarajan, B.: The complexity of fine motion planning. IJRR **7**(2) (1988) 36–42

[45] Erdmann, M.: Using backprojections for fine motion planning with uncertainty. IJRR **5**(1) (1986) 19–45

[46] Smallwood, R., Sondik, E.: The optimal control of partially observable markov processes over a finite horizon. Operations Research **21**(5) (1973) 1071–1088

[47] Canny, J., Reif, J.: New lower bound techniques for robot motion planning problems. In: IEEE Symposium on Foundations of Computer Science. (October 1987)

[48] Ross, S., Pineau, J., Paquet, S., Chaib-Draa, B.: Online planning algorithms for pomdps. JAIR (2008)

[49] Kurniawati, H., Hsu, D., Lee, W.S.: Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: RSS. (2008)

[50] Bai, H., Hsu, D., Kochenderfer, M., Lee, W.S.: Unmanned aircraft collision avoidance using continuous-state pomdps. In: RSS. (June 2011)

[51] Somani, A., Ye, N., Hsu, D., Lee, W.: Online pomdp planning with regularization. In: NIPS. (December 2013)

[52] Silver, D., Veness, J.: Monte-carlo planning in large pomdps. In: NIPS. (December 2010)

[53] Seiler, K., Kurniawati, H., Singh, S.: An online and approximate solver for pomdps with continuous action space. In: ICRA. (May 2015)

[54] Koval, M., Pollard, N., Srinivasa, S.: Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In: RSS. (July 2014)

[55] Horowitz, M., Burdick, J.: Interactive non-prehensile manipulation for grasping via pomdps. In: ICRA. (May 2013)

[56] Koval, M., Hsu, D., Polland, N., Srinivasa, S.: Configuration Lattices for Planar Contact Manipulation Under Uncertainty. In: WAFR. (December 2016)

[57] MacDonald, R.A., Smith, S.L.: Reactive Motion Planning in Uncertain Environments via Mutual Information Policies. In: WAFR. (December 2016)

[58] Agha-mohammadi, A.a., Chakravorty, S., Amato, N.M.: Firm: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. IJRR (2013)

[59] Hoerger, M., Kurniawati, H., Bandyopadhyay, T., Elfes, A.: Linearization in Motion Planning under Uncertainty. In: WAFR. (December 2016)

[60] Levine, S., Wagener, N., Abbeel, P.: Learning contact-rich manipulation skills with guided policy search. In: ICRA. (May 2015)

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

[62] LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. IJRR **20**(5) (2001) 378–400

[63] Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. IJRR **30**(7) (June 2011) 846–894

[64] Li, Y., Littlefield, Z., Bekris, K.E.: Asymptotically optimal sampling-based kinodynamic planning. IJRR **35** (April 2016) 528–564

[65] Roy, N., Prentice, S.: The belief roadmap: Efficient planning in belief space by factoring the covariance. IJRR **28**(11-12) (2009) 1448–1465

[66] Bry, A., Roy, N.: Rapidly-exploring random belief trees for motion planning under uncertainty. In: ICRA. (May 2011)

[67] Alterovitz, R., Simon, T., Goldberg, K.: The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In: RSS. (June 2007)

[68] Luo, Y., Bai, H., Hsu, D., Lee, W.S.: Importance Sampling for Online Planning under Uncertainty. In: WAFR. (December 2016)

[69] Littlefield, Z., Klimenko, D., Kurniawati, H., Bekris, K.E.: The importance of a suitable distance function in belief-space planning. In: ISRR. (September 2015)

[70] Berg, J.V.D., Abbeel, P., Goldberg, K.: Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. In: RSS. (June 2010)

[71] Huynh, V.A., Karaman, S., Frazzoli, E.: An incremental sampling-based algorithm for stochastic optimal contro. In: ICRA. (May 2012)

[72] Platt Jr, R., Tedrake, R., Kaelbling, L., Lozano-Perez, T.: Belief space planning assuming maximum likelihood observations. In: RSS. (June 2010)

[73] Sun, W., van den Berg, J., Alterovitz, R.: Stochastic extended lqr: Optimization-based motion planning under uncertainty. In: RSS. (August 2014)

[74] Davis, B., Karamouzas, I., Guy, S.J.: C-opt: Coverage-aware trajectory optimization under uncertainty. IEEE Robotics and Automation Letters **1**(2) (July 2016) 1020–1027

[75] Lee, A., Duan, Y., Patil, S., Schulman, J., McCarthy, Z., van den Berg, J., Goldberg, K., Abbeel, P.: Sigma hulls for Gaussian belief space planning for imprecise articulated robots amid obstacles. In: IROS. (Nov 2013)

[76] Ponton, B., Schaal, S., Righetti, L.: On the Effects of Measurement Uncertainty in Optimal Control of Contact Interactions. In: WAFR. (December 2016)

[77] Nieuwenhuisen, D., van der Stappen, A.F., Overmars, M.H.: Pushing using compliance. In: ICRA. (May 2006)

[78] Jaillet, L., Cortes, J., Simeon, T.: Sampling-Based Path Planning on Configuration-Space Costmaps. IEEE Transactions on Robotics **26**(4) (August 2010) 635–646

[79] Berenson, D., Simeon, T., Srinivasa, S.S.: Addressing cost-space chasms in manipulation planning. In: ICRA. (2011)

[80] Hart, P., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics **4**(2) (1968) 100–107

[81] Hallam, C., Harrison, K., Ward, J.: A Multiobjective Optimal Path Algorithm. Digital Signal Processing **11**(2) (April 2001) 133–143

[82] Iehl, R., Cortés, J., Siméon, T.: Costmap planning in high dimensional configuration spaces. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). (2012)

[83] Devaurs, D., Siméon, T., Cortés, J.: Enhancing the Transition-based RRT to Deal with Complex Cost Spaces. In: ICRA. (2013)

[84] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. (2009)

[85] Diankov, R., Kuffner, J.: Openrave: A planning architecture for autonomous robotics. Technical report, Robotics Institute, Pittsburgh, PA (2008)

[86] Coumans, E.: Bullet 2.73 Physics SDK Manual (2010)

[87] Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: STOMP: Stochastic Trajectory Optimization for Motion Planning. In: ICRA. (2011)

[88] Chen, L., Rong, Y.: Linear time recognition algorithms for topological invariants in 3D. In: International Conference on Pattern Recognition. (2008)

[89] Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine **19**(4) (December 2012) 72–82

[90] Goldman, R.P., Boddy, M.S.: Expressive planning and explicit knowledge. In: Artificial Intelligence Planning Systems. (May 1996)

[91] Sneath, P.H.A., Sokal, R.R.: Numerical taxonomy: the principles and practice of numerical classification. Freeman (1973)

[92] Asafi, S., Goren, A., Cohen-Or, D.: Weak convex decomposition by lines-of-sight. Computer Graphics Forum **32**(5) (2013) 23–31

[93] Denny, J., Sandstrom, R., Bregger, A., Amato, N.M.: Dynamic Region-biased Rapidly-exploring Random Trees. In: WAFR. (December 2016)

[94] Koenig, N., Howard, A.: Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In: IROS. (September 2004)

[95] McMahon, T., Thomas, S., Amato, N.M.: Sampling-based motion planning with reachable volumes: Theoretical foundations. In: ICRA. (May 2014)

[96] Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. IEEE Computational Science & Engineering **5**(1) (1998) 46–55

[97] Ichnowski, J., Alterovitz, R.: Fast Nearest Neighbor Search in SE(3) for Sampling-Based Motion Planning. In: WAFR. (August 2014)

[98] Saurav Agarwal, A.T., Chakravorty, S.: Motion Planning for Active Data Association and Localization in Non-Gaussian Belief Spaces. In: WAFR. (December 2016)

[99] McConachie, D., Berenson, D.: Bandit-Based Model Selection for Deformable Object Manipulation. In: WAFR. (December 2016)

[100] Nguyen, B., Trinkle, J.C.: dvc3D: a three dimensional physical simulation tool for rigid bodies with contacts and Coulomb friction. In: Joint International Conference on Multibody System Dynamics. (August 2010)

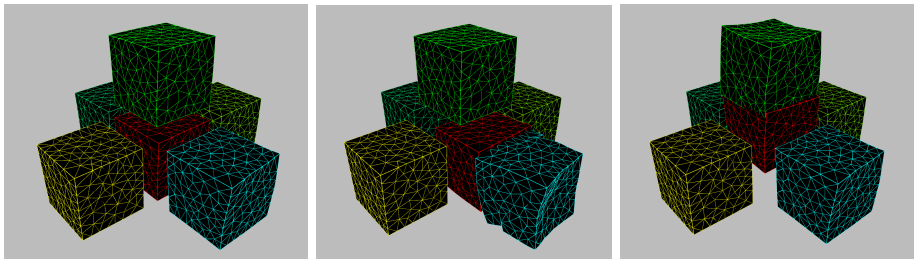
[101] Drumwright, E., Shell, D.A.: An Evaluation of Methods for Modeling Contact in Multibody Simulation. In: ICRA. (May 2011)

Chapter 8

Appendix

Appendix A Learning deformability parameters

While learning physical properties of deformable objects is not a focus of our work, it is useful to ground the deformability parameter used in our models against real-world measurable behavior. Since we have reduced the physical behavior of objects to a single parameter in the range $[0, 1]$, it is not directly equivalent to known physical characteristics of the material (ex. Young's modulus). Similarly, a modelled object might consist of multiple materials, and thus the overall behavior is a combination of the materials involved. Instead, our deformability parameter is primarily a *relative* value between the objects being modelled: two objects that deform in similar amounts when exposed to similar contacts should receive similar deformability values. To demonstrate that this deformability parameter can be recovered using an active manipulation approach like [29, 15, 30], we have performed a simple experiment using the same physics simulator used to recover sensitivity parameters.



(a) (b) (c)

Figure 8.1: (a) The experiment used to demonstrate recovery of deformability parameters. The center red object has known deformability; other objects' are unknown. (b,c) The known object is used to deform the unknown objects, and the deformability of the unknown object is recovered based on the object's deformation.

	Object				
	1	2	3	4	5
Ground truth	0.75	0.625	0.5	0.375	0.25
Recovered	0.813	0.604	0.482	0.413	0.191

Table 8.1: Comparison between known ground-truth and recovered deformability parameters.

In our experiment setup, as shown in Figure 8.1, a set of deformable objects with varying (unknown) deformability are arranged around a central object (in red) with known deformability. In turn, the central object is brought into contact with each of the unknown objects until both objects deform. We then record the final deformed volume of the objects, from which we compute the ratio of volume change. By multiplying this ratio with the known deformability of the central object, we estimate the deformability of the unknown object. We performed this test with 30 iterations for each unknown object, averaging the deformed volume to minimize simulator-induced noise. We can then compare the estimated deformability values to the ground truth deformability values used to configure the simulator. As shown in Table 8.1, we produce a reasonable estimate of the deformability of each unknown object. Note that due to simulation artifacts and noise, we do not expect to recover these values exactly; however, we recover parameters with the correct relative ranking.

Appendix B Fast kinematic simulation

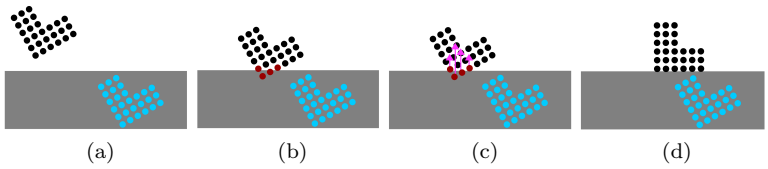


Figure 8.2: The collision resolution process used by our lightweight simulator. From left to right, (a) a robot represented by points (black) moving towards a target (light blue) and an obstacle (gray) (b) collides with the object, triggering the collision resolution. (c) point corrections Δp_n for each colliding point of the robot are computed from the surface normals of the object and applied (d) so the robot complies to produce an in-contact state.

Our approach for planning with uncertainty in Chapter 5 relies on a efficient simulator for the behavior of a controlled compliant robot. While progress has been made in the performance and accuracy of dynamic simulators [100, 101], we require a simulator capable of evaluating hundreds, if not thousands, of robot motions per second to grow the planner's tree in reasonable time. To improve computational performance, we limit ourselves to kinematic simulation, though

our planning framework is agnostic to the simulator being used. Kinematic simulation is only an approximation of true robot motion; however, we mitigate the discrepancy between simulated and real dynamic behavior using policy adaptation to update the planned policy with the results of real-world executions. For our work, the robot is controlled via PD feedback controller with gains K_p, K_d; for error $e_q = q_{desired} - q$ the resulting control input is $\Delta q = K_p e + K_d \dot{e_q}$; however, different controllers can be used with the simulator. For a fixed time limit $t_{simulate}$, we forward simulate the motion of the robot from the current configuration q_t to the next configuration q_{t+1} using the equation below.

$$e_q = q_{target} - q_t \tag{8.1}$$

$$\Delta q = K_p e_q + K_d \dot{e_q} \tag{8.2}$$

$$q_{t+1} = q + \Delta q + \mathbf{F}(\Delta q) \tag{8.3}$$

$$q'_{t+1} = \textsc{ResolveCollisions}(q_{t+1}) \tag{8.4}$$

Collision resolution and robot compliance are modelled by ResolveCollisions, which iteratively corrects colliding configurations q_{t+1} until an in-contact configuration q'_{t+1} is reached. For performance purposes, the environment E is modelled using a voxel grid that stores the surface normals for all obstacles in E, and the robot R is modelled using a set of points for each link. Collision checking of a configuration q is performed by transforming every point of the robot into the environment and checking if any of the corresponding voxels belong to an obstacle. If any voxels belong to an obstacle, the collision is resolved by iteratively applying corrections Δq as shown in Figure 8.2. Each correction Δq is the product of the individual point corrections Δp_n for each colliding point, where Δp_n is the product of the surface normal of the collided obstacle and penetration distance of point p_n, and the Jacobian $\mathbf{J}$ pseudoinverse of the robot for configuration q and point p_n as shown below:

$$\Delta q = \mathbf{J}(q, p_1, p_2, ...)^{+}[\Delta p_1^T, \Delta p_2^T, ...]^T \tag{8.5}$$

Intuitively, this computes the change in configuration necessary to move points $p_1, p_2, ...$ out of collision, where the correct direction to move of out of collision is approximated by the surface normal of the collided object. Note that this approximation is only valid if the maximum penetration of an obstacle is small; thus we apply the motions computed by Equation 8.1 and Equation 8.5 over small timesteps to ensure that the workspace motion of the robot is small. To avoid oscillation in confined spaces where corrections from one set of contacts may produce new contacts, an adaptive step scaling mechanism is necessary to detect oscillations and reduce step sizes accordingly. Importantly, to avoid over-determining Equation 8.5, we must ensure that penetration distances are correctly computed; simply estimating using a signed-distance field will produce incorrect behavior, as penetration distances are discretized by the field. Similarly, using the gradient of a signed-distance field to supply surface

normals will result in unexpected behavior on edges and corners, where the direction of robot motion determines which face of the object is contacted, and thus the surface normal. This issue is avoided by precomputing true surface normals for every contacting face of the voxels of the environment, and then retrieving the surface normal corresponding to the robot's direction of motion.

To extend the environment-collision resolution of Equation 8.5 to self-collisions of a multi-link robot, we identify colliding points on the robot's links and generate per-point corrections using a pure momentum-based model of general n-body inelastic collisions, as this method does not require full dynamics. We approximate the momentum of colliding links using the mass of the link and the Δq of the last simulated step of the controller. Of course, since this approach does not account for the dynamics of the robot, it will exaggerate the motion of near-root links as a result of self-collision. To mitigate this error, we approximate the inertia of the arm by assigning $mass_i$, mass of link i, to $mass_i = mass_i + mass_{i+1} + ... + mass_n$ while evaluating the resulting velocities from the inelastic collision. These resulting velocities are used directly as point corrections Δp_n.

While our kinematic simulation does not consider surface friction which could hamper sliding motions, we address this using a simple controller discussed in Appendix C and our simulation results show that this limitation does not overly impair the performance of our planner, though unexpected jamming could still occur.

Appendix C Execution controllers

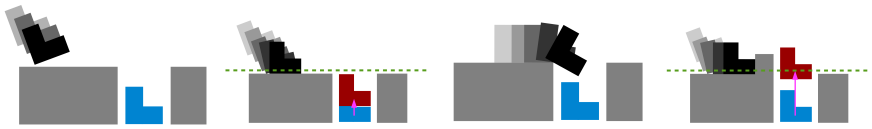


Figure 8.3: Our contact motion controller helps mitigate the effects of contact friction. (a) The robot approaches contact while moving towards the goal in blue. (b) The robot makes contact and becomes stuck on the surface, from which we estimate a plane (green) that locally approximates the surface and adjust the goal by ϵ_{adjust} shown in magenta to reduce contact force until (c) the robot resumes moving. (d) Alternatively, the robot remains stuck for i iterations until $i\epsilon_{adjust} = \epsilon_{adjust}max$ and the controller terminates.

We use the Gazebo dynamics simulator in both planar and 3D environments to simulate execution of robot motion including contact with obstacles. In the kinematic simulator used during planning, a PD position controller attempts to reach a target configuration q_{target} by commanding velocities to the robot. Likewise, we control the simulated robot in Gazebo using a position controller

that receives q_{target} and commands velocities. To safely achieve those velocities in collision and contact, a velocity controller commands forces and torques that move the simulated robot. Unlike the kinematic simulator, which ignores friction and dynamic effects to achieve faster runtime, the dynamic simulator incorporates friction between the robot and the environment. To mitigate the effects of friction in execution, we use a *contact motion controller* illustrated in Figure 8.3 which adjusts q_{target} to reduce contact forces that cause the robot to become stuck.

When the contact motion controller receives a new target position, it first commands q_{target} without modification. For the duration of execution t_{exec}, at each iteration the controller records the trajectory of the robot and checks if the robot has become stuck, i.e., if the total motion over a sliding window of the trajectory is below a threshold ϵ_{stuck}. If the robot is stuck, we assume that the surface on which the robot is stuck can be locally approximated as a plane, which we can estimate from the recent motion of the robot. Once the robot is stuck, the controller then fits a plane defined by point P_{plane} and normal vector $\overrightarrow{N_{plane}}$ to the sliding window of the trajectory and projects q_{target} towards the plane:

$$q'_{target} = q_{target} + (\frac{\overrightarrow{q_{target}, P_{plane}} \cdot \overrightarrow{N_{plane}}}{\overrightarrow{N_{plane}} \cdot \overrightarrow{N_{plane}}} \overrightarrow{N_{plane}})(i\epsilon_{adjust}) \tag{8.6}$$

Here, on the ith stuck iteration of the controller (i.e., the robot has be stuck for i consecutive iterations of the controller), the controller computes $\overrightarrow{q_{target}, P_{plane}}$, the vector from q_{target} to P_{plane}, and projects it onto $\overrightarrow{N_{plane}}$ to compute an adjustment vector. The target configuration is then moved along the adjustment vector towards the plane by $i\epsilon_{adjust}$, where ϵ_{adjust} is the amount to adjust at each step. If the controller exceeds the time limit t_{exec} or $i\epsilon_{adjust} \geq \epsilon_{adjust}max$, the controller reports that the robot has become "completely stuck". Intuitively, this controller reduces contact forces (and thus the effects of friction) by moving q_{target} towards the surface of the obstacle approximated by fitting a plane to the trajectory. If the robot continues to move, or if the robot resumes moving after being stuck, the controller commands the original q_{target}. For both $SE(2)$ and $SE(3)$ simulation tests, we used $\epsilon_{adjust} = 0.01$ and $\epsilon_{adjust}max = 1.0$ (i.e., it will attempt 100 stuck iterations before terminating the motion). In testing in both Gazebo and with the cartesian impedance controllers of the Kuka IIWA robot, the contact motion controller enables reliable sliding motion.

A structurally similar contact motion controller is used with the Baxter robot; however, instead of fitting a plane in R^7 and projecting the target towards it, we use the kinematic simulator described in Appendix B to predict the next adjusted target configuration. At a stuck configuration q_{stuck}, we forward-simulate using the kinematic simulator towards q_{target} for a brief timestep, and record the resulting configuration $q_{simulated}$. We then interpolate between q_{target} and $q_{simulated}$ by $i\epsilon_{adjust}$ to produce the new adjusted target q'_{target}.