Relatively Robust Grasping

by

Kaijen Hsiao

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author .....
Department of Electrical Engineering and Computer Science
September 4, 2009

Certified by .....
Tomás Lozano-Pérez
Professor
Thesis Supervisor

Certified by .....
Leslie Kaelbling
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Relatively Robust Grasping

by

Kaijen Hsiao

Submitted to the Department of Electrical Engineering and Computer Science
on September 4, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

This thesis presents an approach for grasping objects robustly under significant positional uncertainty. In the field of robot manipulation there has been a great deal of work on how to grasp objects stably, and in the field of robot motion planning there has been a great deal of work on how to find collision-free paths to those grasp positions. However, most of this work assumes exact knowledge of the shapes and positions of both the object and the robot; little work has been done on how to grasp objects robustly in the presence of position uncertainty. To reason explicitly about uncertainty while grasping, we model the problem as a partially observable Markov decision process (POMDP). We derive a closed-loop strategy that maintains a belief state (a probability distribution over world states), and select actions with a receding horizon using forward search through the belief space. Our actions are world-relative trajectories (WRT): fixed trajectories expressed relative to the most-likely state of the world. We localize the object, ensure its reachability, and robustly grasp it at a specified position by using information-gathering, reorientation, and goal-seeking WRT actions. This framework is used to grasp objects (including a power drill and a Brita pitcher) despite significant uncertainty, using a 7-DOF Barrett Arm and attached 4-DOF Barrett Hand equipped with force and contact sensors. Our approach is generalizable to almost any sensor type, as well as wide ranges of sensor error and pose uncertainty.

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor

Thesis Supervisor: Leslie Kaelbling
Title: Professor

Acknowledgments

I would like to thank all the people who made MIT a happy and comfortable place for me during my time in grad school.

First, I would like to thank Tomas and Leslie, my advisors, for being incredibly understanding, supportive, patient, and helpful. Each person's grad school experience depends heavily on who his or her advisors are, and many a time I've counted my lucky stars that I have advisors as wonderful as these. I don't think there are many other advisors out there who will grant you the freedom and space to do what you want to do, and at the same time, work closely with you when you need it, writing code and papers for you and even revising your writings while on vacation. I'm sure as soon as I leave MIT, I'll miss being under their sheltering wings.

I would like to thank all the other people who have helped me with my research over the years: Ross Glashan, an incredibly talented staff member who was in our group for one gloriously productive and far-too-short year, and also all the UROPs who have helped me over the years. I would like to thank Huan Liu in particular, who has been my eager, helpful, and companionable sidekick for the past year and a summer, and who gets to inherit this entire mess when I leave.

I would like to thank my officemates in G585, for providing a fun and social environment in which I felt at home, during the few-and-far-between times in which I was actually in my office.

I would like to thank the helpful and kind people at Barrett Corp., for providing lots of assistance and repairing my robot over and over again, and even driving to Stata for in-house visits and to drag pieces of my robot away and bring them back again.

I would like to thank Nick Roy, my third committee member, for many helpful comments on my thesis.

I would like to thank my family: my parents, who drove to Cambridge many a weekend to bring lunch and food supplies for me and apple branches and veggies for my bunnies, and who made it possible for me to get to this point in the first place; and my brother, who has, amusingly enough, been at MIT the entire time I've been here, and who is always willing to drive me someplace when I call whining about feeling stir-crazy.

Last, but most certainly not least, I would like to thank my sweet and wonderful husband Michael, for being the light of my life and the stars in my sky.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction and Related Work

Robots are highly adept at manipulation in carefully controlled environments, where the locations and shapes of things are exactly known. For instance, manufacturing robots can put together cars faster and more accurately than any human. However, in unstructured domains such as peoples' homes, robots that attempt to manipulate objects today (outside of carefully controlled demonstrations) are usually still fairly clumsy, often knocking things over or dropping them. In order to create robots that can autonomously assist humans in unstructured environments–for instance, helpers for the elderly or disabled, or even just robot maids–we need to figure out how to increase their dexterity.

Many motion planning techniques have been developed to allow robots with many degrees of freedom to plan complex motions through free space [27, 31]. Grasp planning has advanced to the point where if the locations and shapes of the robot, the environment, and the objects we wish to manipulate are exactly known, we can usually figure out where to stably grasp said objects [38]. Unfortunately, there is always uncertainty in the world: sensors are noisy, objects are never quite where you think they are or exactly the shape you expected, and robots are difficult to calibrate exactly. Because of these types of uncertainties, traditional open-loop strategies for manipulating objects are not sufficiently robust. Accurate sensors and careful cali-

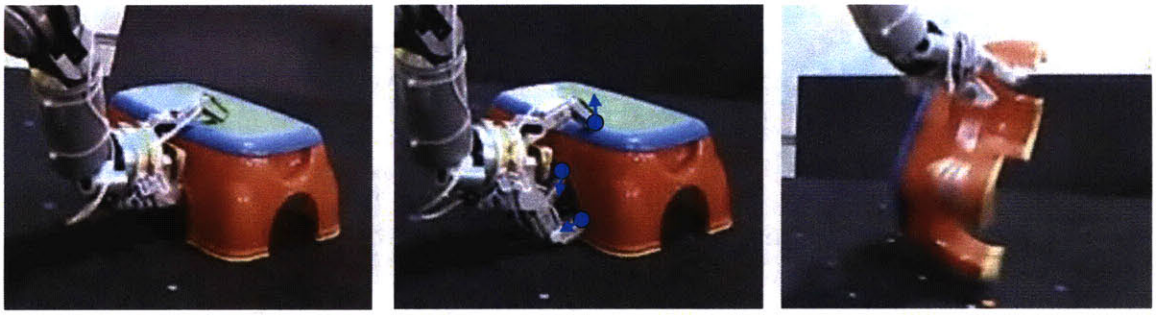

Figure 1-1: A grasp of a stool fails due to unstable contacts that are not in the desired locations.

bration can increase robustness, but the configuration of the robot and the objects in the world can never be exactly known. If we can model the uncertainty in the world and reason explicitly about it, we can give our robots the ability to figure out when their actions are likely to succeed or fail, to gather information as needed, or to recover from problems in an intelligent way.

It is useful to distinguish between modes of uncertainty that can be effectively modeled, and those that cannot. In situations with unmodelable uncertainty, such as insertion of keys into locks, very fine-grained details of the surfaces can have large effects on the necessary directions of applied forces, and the available sensors can gain little or no information about those surfaces. When the uncertainty is unmodelable, we must fall back to strategies such as "wiggling" the key, which are highly successful without ever building a model of the underlying situation.

Modelable uncertainty, on the other hand, typically occurs at a coarser scale. In attempting to pick up the stool in Figure 1-1, for example, a robot with vision might recognize the object type and even have a reasonable model of the stool, but still have significant remaining uncertainty about the pose or exact shape of the object. A typical strategy to use in this case is to assume that the estimated pose and shape are correct, plan an appropriate grasp, and attempt to execute the grasp open-loop. However, due to small errors in those estimates, the grasp can end up knocking the object over, missing entirely, or as in this case, merely making contacts that are different than those that were planned. In this example, even though the contacts made while executing the grasp open-loop are sufficient to pick up the stool, one finger is on the edge of the side of the stool, and thus the grasp is unstable enough that the robot drops the stool in mid-air after picking it up off the table.

There are many previously-used methods of attempting to do better than just grasping once, open-loop.

- The first, and perhaps most obvious, is to detect failure and try again to grasp, which is sufficient in many situations. However, if the first grasp knocks the object over or moves it out of range, a second try may not be possible. Even if the first grasp does not change the situation at all, if the robot does not reason about why it failed, a second try may not do any better than the first.

- Visual servoing is a popular way of executing a grasp in a closed-loop fashion [24], in which the location of an object relative to the robot is tracked by one or more cameras (often eye-in-hand, but not always), and the robot continuously adjusts its position while moving to grasp it. These methods can be particularly useful for the initial grasp approach, but they rely on the relevant visual features remaining visible to the camera(s) throughout the grasp in order to continue to work, which is often difficult when the hand is close to the object and each is likely to obscure parts of the other.

- Another possible way to make grasps more robust is to hand-code object-specific policies. For instance, in [34], a hand-coded sequence of robust control primitives was used to enable Robonaut to grasp a powerdrill. This can work well

if there are few objects that the robot will ever encounter, but a great deal of human effort is needed to enable the robot to be able to grasp each new object.

- If a specific grasp is not needed, it is possible to make a robot use tactile sensing to adjust an initial grasp until it is locally stable [45]. Using simple assumptions about the local shape of the object under the sensed contacts, the robot can displace its grasp incrementally to minimize the net force and torque of the grasp. This is an excellent method to use in the absence of an object model, but it is vulnerable to local minima; also, the force/torque minimum may not be a stable grasp, even if a good grasp is possible elsewhere on the object.

- Along the same lines, one can also use "pre-touch" sensors such as IR or electric field sensing with simple, reactive control laws to adjust the position of the hand relative to the surface of an object prior to grasping, the goal being to center the local part of the object within the hand and to cause all fingers to touch the surface at the same time [23, 35]. Although these methods can add a fair amount of robustness in certain situations, such local adjustments also do not necessarily result in a stable grasp. Also, each sensor type is unable to deal with certain types of objects. IR sensors do not see shiny or transparent surfaces easily, and electric field sensors do not see thin, dielectric objects.

- Finally, one can use tactile sensing along with a fixed, hand-selected sequence of information-gathering actions to exactly locate the exact position of an object before grasping, as in [42]. However, without being able to select appropriate actions on-line as needed, it can be difficult to generalize to a large range of situations without being overly cautious.

To motivate the methods used in this thesis, we make the following observation: when grasping the stool in Figure 1-1, if the robot had had tactile sensors, it might have seen the contacts shown in the middle panel of Figure 1-1 while attempting the open-loop grasp. Given the observed contacts, and a reasonable model of the object shape, the robot should have been able to figure out that the grasp that was made was not the desired one. At that point, it could have used its model of the situation to choose actions that would both gather more information and make progress toward the desired grasp.

1.1 Belief-based programming

In order to choose such actions, we can use "belief-based" programming, in which we express what we know about the uncertain aspects of the world as a "belief state", which is a probability distribution over possible world states, and then select actions based on the current belief state.[1]

[1]Probabilistic belief states are one type of "information state." It is possible to reason in non-probabilistic information spaces (for example, lists of possible contacts). However, probabilities place "weights" on information states and can sometimes lead to more efficient decisions.

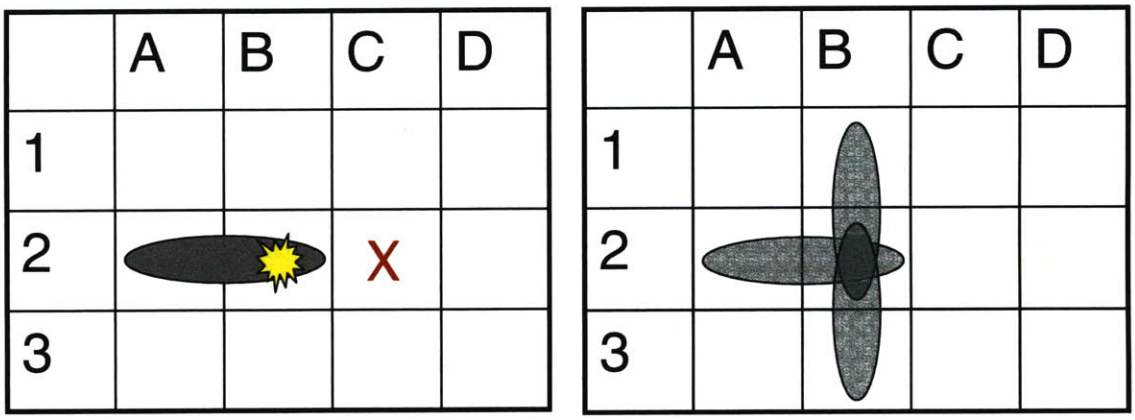

Figure 1-2: An example of a belief state in a tiny battleship game. The grey oval in the left grid represents the actual location of a two-block ship (or, equivalently, object on a table). After observing a 'hit' at B2 and a 'miss' at C2, we can represent our beliefs about where the ship is as a probability distribution, as shown on the right.

As a concrete example of how this might look, we can consider the simple example of a tiny, one-sided game of Battleship. In this tiny Battleship game, there is only one ship that is two blocks long, and we just have to figure out where it is in a grid that is 3 blocks by 4 blocks, as shown in Figure 1-2. The ship is actually in grid squares A2 and B2, but all we know at the start of the game is that it is somewhere on the grid. We might guess C2, and observe a 'miss', and then guess B2, and observe a 'hit'. We can express what we know about the world at this point in the form of a probability distribution over the possible positions of the ship, which is our belief state. The belief state after these two moves is shown in the right side of Figure 1-2; there remain three possible positions for the ship, each with probability 1/3.

The ship in our example could just as easily be a box sitting on a table that we have to grasp, with high enough starting uncertainty (from a vision system that gives poor estimates of the location of the box) that we need to use our tactile sensors to locate the box before we can grasp it. The main difference in this case would be that the box is unlikely to be in only a few discrete locations on the table. Also, unlike ships in the game of Battleship, touching the box to gather information can cause it to move.

Given a belief-state representation of the world, we would like to be able to update the current belief state based on the action we just took, and the observation we received upon taking the action. We would also like to be able to automatically select appropriate actions to gather information and/or to reach our goals, based on the current belief state. These two modules are called a state estimator and a policy, as shown in Figure 1-3. The state estimator (SE) recursively computes a belief state, as a function of the previous belief state, action and observation, and is usually relatively easy to construct. The problem of finding a good policy (π), that selects appropriate actions based on the current belief state, is much more difficult. In some

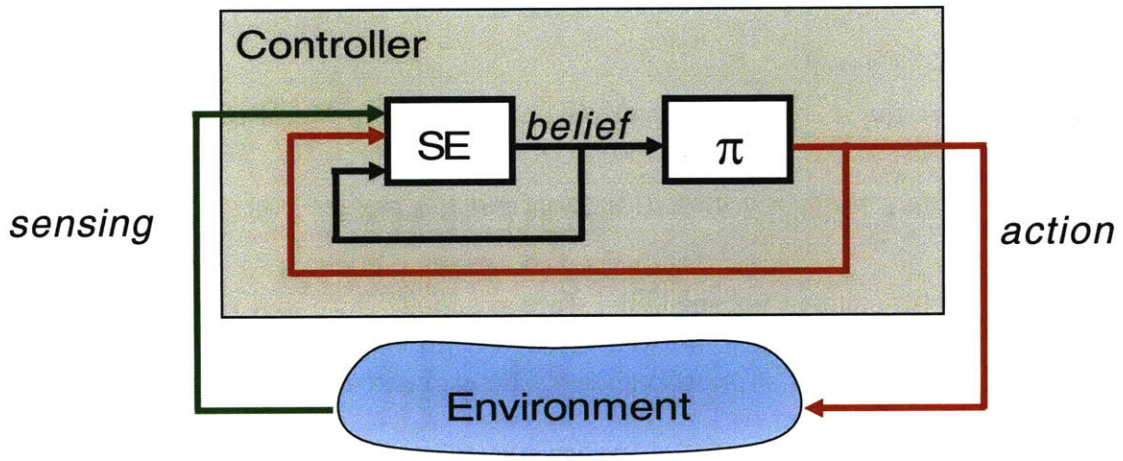

Figure 1-3: A POMDP controller consists of a state estimator (SE), which computes an updated belief state as a function of the previous belief state, action, and observation, and a policy (π), which selects actions based on the current belief state.

cases, human programmers can write policies directly; but as domains become more complex, we would like to be able to generate policies automatically, given a model of sensing and action dynamics.

1.2 Partially observed Markov decision processes (POMDPs)

POMDPs [50] are the primary model for formalizing decision problems under uncertainty. If we can express our problem as a POMDP, we can use existing algorithms for automatically finding policies for problems described as POMDPs. A POMDP model consists of finite sets of states S, actions A, and observations O; a *reward* function $R(s, a)$ that maps each underlying state-action pair into an immediate reward; a *state-transition model* $P(s'|s, a)$ that specifies a probability distribution over the resulting state s', given an initial state s and action a; and an *observation model* $P(o|s)$ that specifies the probability of making an observation o in a state s. Given the model of a POMDP, the problem of optimal control has the two parts described earlier, in Figure 1-3: state estimation and policy execution.

State estimation, otherwise known as belief-state updating, is a straightforward instance of a Bayesian filter. The robot's current belief state is an n-dimensional vector, b_t, representing $\Pr(s_t|o_1 \ldots o_t, a_1 \ldots a_{t-1})$, a probability distribution over current states given the history of actions and observations up until time t.

Given a new action a_t and an observation o_{t+1}, the new belief state b_{t+1} is given

by

$$\begin{aligned}
&\Pr(s_{t+1} = i | o_1 \ldots o_{t+1}, a_1 \ldots a_t) \\
&\quad \propto \sum_j \Pr(s_t = j | o_1 \ldots o_t, a_1 \ldots a_{t-1}) \Pr(s_{t+1} = i | s_t = j, a_t = a) \Pr(o_{t+1} = o | s_{t+1} = i) \\
&\quad = \sum_j b_{t_j} \Pr(s_{t+1} = i | s_t = j, a_t = a) \Pr(o_{t+1} = o | s_{t+1} = i)
\end{aligned}$$

where b_{t_j} is vector element j of the belief state at time t. Note that the first factor is an element of the state transition model and the second is an element of the observation model. The constant of proportionality is determined by the constraint that the elements of b_{t+1} must sum to 1.

The problem of deriving an optimal policy is much more difficult than state estimation. The policy for a POMDP with n states is a mapping from the $n-1$-dimensional simplex (the space of all possible belief states) into the action set. Although a policy specifies only the next action to be taken, the actions are selected in virtue of their long-term effects on the agent's total reward. Problems that are naturally described as having a goal state can be encoded in this framework by assigning the goal states a high reward and all the rest zero; but an advantage of a more general reward function is that it can easily also penalize other conditions along the way, or assert two goal regions, one of which is more desirable than the other, and so on.

Generally, we seek policies that choose actions to optimize either the expected total reward over the next k steps (finite-horizon) or the expected infinite discounted sum of reward, in which each successive reward after the first is devalued by a discount factor of γ.

These policies are quite complex because, unlike in a completely observable MDP, in which an action has to be specified for each state, in a POMDP, an action has to be specified for every *probability distribution* over states in the space. Thus, the policy will know what to do when the robot is completely uncertain about its state, or when it has two competing possibilities, or when it knows exactly what is happening.

A more detailed introduction to POMDPs and various methods for doing action selection in POMDPs can be found in [8].

1.3 Computing optimal policies off-line

Because of the special properties of POMDPs, the optimal finite-horizon policy is always finitely representable as a linear partition on the underlying belief space, with an action associated with each region in the partition [50]. This property often, though not always, holds of optimal infinite-horizon discounted policies, as well. Nonetheless, the exact optimal policy for a POMDP can be very difficult to compute. In general, the time to solve the finite-horizon problem is doubly exponential in the horizon, and the infinite-horizon problem is undecidable.

Although computing the optimal policy is generally intractable, it is often possible to derive good approximate solutions by taking advantage of the fact that the set of

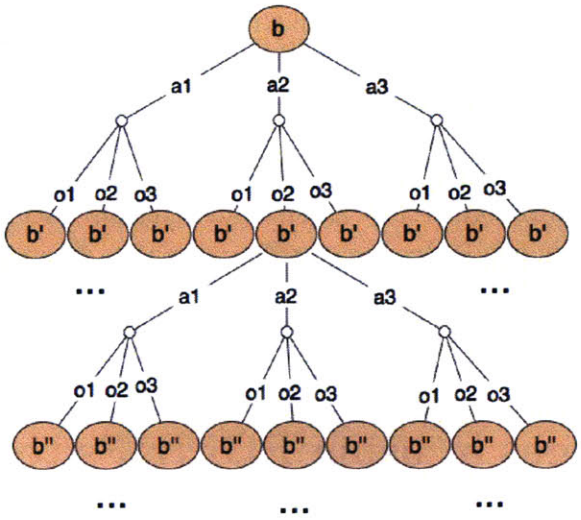

Figure 1-4: The tree-structure of the POMDP search space.

states that are reachable under a reasonable control policy is typically dramatically smaller than the original space. Thus, many recent approximate solvers such as [43, 52, 51, 26] that concentrate on sampling belief states generated by trajectories through the space can be very efficient and effective. Recently, [4] has applied the Perseus method [52] for efficiently generating approximate POMDP solutions to a practical application with more than 20 million states and 20 actions.

Nonetheless, even deriving good approximate solutions in this fashion is limited to problems with small goal horizons and low action and observation branching factors. This can be most easily seen by visualizing the execution of a POMDP as a tree, as shown in Figure 1-4. A POMDP policy is given the initial belief state at the start of execution, and has to pick an action. Each action, when executed in the world, can result in a set of possible observations, and each action and observation pair can be used to generate a new belief state via the belief state update. From the new belief state, the policy must again pick an action, and so on. We can see that the size of the reachable belief space, which is the same as the size of the POMDP tree, is exponential in the goal horizon (how many steps it takes to reach the goal), and polynomial in both the number of actions and observations at each branch. Thus, even though sampling belief states in the reachable belief space is vastly more efficient than sampling the entire belief space, the space of POMDP models for which an approximately optimal policy can be solved off-line in a reasonable amount of time is still quite limited.

1.4 Forward Search Methods

When using POMDPs, we need to be able to choose an action for any belief state we might encounter, and solving for the optimal policy off-line allows us to do that. However, at any point in time during execution, we only have one current belief state.

At that point, we need not think about all the other possible belief states we could be in; we only need to decide what to do for the belief state we are actually in. Rather than solving for an optimal policy for all belief states up front, we can instead do a forward search from the current belief state to find the action with the best reward after n steps, where n is the search horizon. This method requires more time for planning on-line, as opposed to the much faster action selection that can be done when a stored policy is available, but can be tractable in many situations in which off-line policy generation is intractable. A more detailed explanation of POMDP forward search will be presented in section 3.2.

1.5 Challenges in using POMDPs to model object manipulation

POMDPs are a natural framework for modeling decision problems under uncertainty. However, selecting actions in a POMDP is generally only tractable for certain limited classes of problems, for the reasons described above. When using typical off-line policy solvers, the states, actions, and observations must all be discrete, finite sets that are fairly limited in size. When using forward search, we are somewhat less constrained. Even so, the belief state must be compact to represent and reasonably efficient to update, the action and observation branching factors must be quite low, and the horizon short. Finally, we need to be able to use the observation and transition models for belief update quickly. Unfortunately, the problem of manipulation is inherently continuous, and observation and transition models can be very expensive to compute in this domain, which presents a number of challenges in using POMDPs to select actions.

Belief state representation Objects that one would like to manipulate can be anywhere in a continuous workspace of configurations, not just at a small number of discrete locations. Likewise, the state of a robot arm with many joints is fairly high-dimensional and continuous. With so many dimensions in the relevant state space, trying to do a simple discretization of the space would be neither compact nor amenable to fast belief updates. Furthermore, unlike many problems where a continuous state space can be fairly accurately represented by a single Gaussian, our belief about where an object is after a single failed grasp is typically multi-modal. All of these things can make it difficult to compactly represent a belief state that is sufficiently rich to represent our problem.

Long horizons If we naively choose a small, primitive set of actions such as 'move a small distance in the x-direction' and 'move a small distance in the y-direction', as is often done in mobile robot control problems, then we are likely to need a large number of actions to reach our target grasp position. The number of actions in the goal horizon, h, is also the depth of the POMDP search tree, so the size of the space

that must be explored to find a solution is exponential in h. Thus, we need to choose actions that will get us to the goal in a small number of steps.

Large action branching factor In order to grasp an object, we need to be able to move the robot to where the object is, at the very least. If we assume the object could be anywhere in a fairly large workspace, the space of robot motions that we might need to perform in order to grasp the object, for a robot with many joints, is enormous. However, if the number of actions that one has to consider at each step is a, the size of the POMDP search tree increases as a^h. Thus, in addition to having to select actions that make fast progress towards an arbitrary object position, we also need to limit the number of robot motions that we need to consider at each step in order to do so. These two goals are often in direct opposition, so a careful balance of the two must be found to make action selection tractable.

Large observation branching factor The observations we can make while grasping take the form of sensors that provide continuous or near-continuous information: vision, robot proprioception, tactile sensor locations and normals, and so on. However, the number of observations we need to consider at each step, o, is another branching factor in the POMDP search tree, which means that the size of the search tree also increases as o^h. Thus, we need a small, discrete set of possible observation outcomes to consider at each time step. This can make it difficult to make use of a rich and continuous sensorium.

Computationally complex observation and transition models Predicting what will happen while manipulating objects, in terms of the contacts made between a robot hand and an object, is extremely computationally complex. The difference between being in contact and not being in contact is all-important while trying to manipulate objects, and tiny changes in the positions of the object or robot hand can cause the contacts that are made to change entirely and abruptly. A stable, force-closure grasp shifted by a tiny amount can miss the appropriate surfaces entirely, disrupting the grasp. Furthermore, because we are forced to make contact with objects in order to grasp them, we have to consider the possibilities of objects moving or even tipping over. Thus, unlike with many problems described by POMDPs today, computing the action and observation models for object manipulation requires computationally expensive simulation of a large number of possible outcomes even in the presence of small amounts of uncertainty. This is a problem for action selection in POMDPs, because every branch in the POMDP search tree requires the use of both observation and transition models to do the belief update, and if using these models takes a long time, not many branches can be explored.

1.6 Our approach

The approaches we outline here apply to any domain in which a robot is moving or interacting with other objects and there is non-trivial uncertainty in the configuration.

In this thesis, we concentrate on the illustrative problem of a robot arm and hand grasping objects resting stably on a table, in particular goal locations. We assume that the robot's position in the global frame is reasonably well known, and that we have a reasonable model of the object shape, but that there can be significant uncertainty about the relative pose of the object to be manipulated. Additionally, we assume that there are tactile sensors on the robot that will enable it to perform compliant motions and to reasonably reliably detect when it makes or loses contacts.

Our objective is to robustly grasp objects in particular goal locations, taking actions to gather information as needed to ensure robustness, without having to be overly cautious. By modeling manipulation as a POMDP, we can explicitly reason about the remaining uncertainty, which allows us to estimate how likely it is that we have achieved our goal grasp, and predict the effects of various actions in terms of gathering information and achieving our goals. This thesis will describe two methods of modeling the problem of grasping objects under uncertainty as a POMDP, each of which addresses the issues of having inherently continuous state, action, and observation spaces in a different way.

In our first approach, we express manipulation problems as finite-space POMDPs and use off-line policy generation to select actions. We examine simple problems involving grasping rectilinear objects in two dimensions, and use guarded compliant motions such as "move left until a contact change occurs" as our action set. These actions act as "funnels," causing large sets of states to transition to much smaller sets of states. This allows us to use model minimization methods to create small, discrete sets of abstract states. We can then use approximate off-line policy generation with our abstract model to select appropriate actions.

In our second approach, we switch to expressing manipulation problems as continuous-space POMDPs and use forward search to select actions. We examine the more complex problem of grasping mesh objects in three dimensions, and introduce the concept of a World-Relative Trajectory (WRT), a temporally extended action that consists of an entire grasp trajectory expressed relative to the current most likely pose of the object. WRTs can be viewed as pruning tools that allow us to prune a wide and continuous range of potential robot motions down to only a few motions that are likely to be useful given the current belief state. We select actions on-line using POMDP forward search with a receding horizon. Using forward search also allows us to use continuous-space observations, by only requiring us to consider a small number of observations that we are likely to encounter when executing the relevant actions.

1.7 Related Work

Preimage backchaining An early formulation of the problem of automatically planning robot motions under uncertainty was the preimage backchaining framework [33]. A goal pre-image is a set of configurations in which the robot can reach the goal through a single compliant motion, under bounded errors in sensing and motion. Chaining backwards involves finding the pre-image of the goal preimage (all configurations that can reach the goal in two compliant motions), and so on back towards the

current position of the robot. If the entire range of uncertainty of the current position can be contained in one such chain, then the compliant-motion strategy contained in that chain is guaranteed to reach the goal. Algorithms for computing such strategies are presented in [16] and [28], and the method is extended in [15] to include model (shape) uncertainty, and to generate plans that still have a chance of succeeding when no plan is guaranteed to succeed all the time. Unfortunately, computing preimages is quite difficult and computationally expensive (exponential in the size of the input problem). Also, without reasoning about the probability of outcomes, this type of framework only separates plans into ones that are guaranteed to succeed, that might succeed, or that cannot succeed, with no notion of explicitly gathering information or taking actions that are most likely to succeed.

More recently, both probabilistic and nondeterministic versions of the planning problem through information (belief) space were formulated in [29]. This formulation shares a great deal with POMDP models, and a path planner is presented in [30] that generates policies under this formulation, using dynamic programming and a cost-to-go function (through either state or information space). However, the time complexity of computing the cost-to-go function is exponential in the dimension of the belief space, which makes it intractable for all but the simplest problems.

A direct probabilistic extension of the preimage backchaining framework that shares much with our finite-space POMDP approach, including the idea of constructing an abstract state space by grouping underlying states with similar transition behavior, is described in [7]. They describe how one might compute the necessary probabilistic pre-images, but do not have effective algorithms for doing so, nor do they provide a method for conditioning actions on observations.

A recent extension of preimage backchaining to dynamic tasks by sequential composition of feedback primitives is described in [10].

Contact state strategies There is a substantial body of previous work (for example [5, 32, 49, 36]) that specifies assembly strategies as paths through sequences of contact states, typically a contact between particular surfaces of the assembled objects. The relevant sequences of states to achieve the goal are variously obtained from a human designer or from analysis of a human performing the assembly. The control strategy is then to identify the current state of the assembly from the history of perceived positions and forces [39, 36, 49, 32, 5] and to choose the appropriate next action; this is a problem that can be quite challenging and which has been the subject of a great deal of work.

The finite-space POMDP approach we describe here is similar in that actions are chosen only based on contact observations. However, it advances on this approach in several ways: (a) it provides an automated way of finding the states and the plans through the state space, (b) it does not require unambiguous identifications of the current state of the manipulation but only a characterization of a probability distribution over the states (the belief state) and (c) it provides an integrated way to choose actions that will provide information relevant to disambiguating the state while also achieving the goal.

Reinforcement learning Another approach for dealing with uncertainty in manipulation is to learn optimal control policies through reinforcement learning. In [21], a system is described that learns optimal control policies in an information space that is derived from the changes in the observable modes of interaction between the robot and the object it is manipulating. Similarly, [44] uses reinforcement learning to generate policies for a k-order MDP whose actions are contact relative motions, or motions that are compliant displacements of the hand relative to sensed contacts. Such methods can work well for simple problems such as grasping basic shapes, but they require a great deal of training data, and do not generalize well to complex problems.

MDP models of manipulation An intermediate position is to assume that there is uncertainty in the outcomes of actions, but that the uncertainty will immediately be resolved through observations. Alterovitz et al.[2] construct and solve such an MDP model for guiding non-holonomic needles. If the current state is roughly observable and the uncertainty is all in the outcomes of actions, this can be an excellent method to use.

Belief-state estimation Belief-state estimation is a central part of most probabilistic approaches to control and has become standard in mobile-robot control [55]. Although it is much less common in the manipulation literature, several examples exist (e.g., [42, 18]).

Finding fixed strategies most likely to succeed under uncertainty There has been a great deal of recent work on generalizations of the motion planning problem that take positional uncertainty and the potential for reducing it via observations into account and plan trajectories through the space that will maximize the probability of success or related other objectives (e.g., [46, 20, 13, 37]). Burns et al. [9] have a different model, in which the system selectively makes observations to reduce uncertainty during planning, but the resulting plan is executed open-loop. Erickson et al. use "active localization" in [17] to localize a blind mobile robot with a contact sensor within a known map from the expected contacts that result from "move-until-contact" commands. They maintain a belief state, and select actions that gather information by using entropy as a heuristic for picking among actions. However, they also plan off-line for a fixed action sequence that is likely to succeed, rather than using on-line belief updates and a contingent policy to adjust to on-line outcomes.

1.7.1 POMDPs for mobile-robot navigation

There is a long history of applying POMDPs to mobile-robot navigation, beginning with simple heuristic solution methods [12, 48], then applying more sophisticated approximations [52, 43, 47, 51, 54]. The problem of manipulation can be loosely viewed as trying to navigate a robot to a particular location on an object. However, in manipulation, contact is not only unavoidable but is actually necessary, and so reasoning

about outcomes has to include reasoning about different contact conditions. In navigation, one typically only reasons about staying within free space versus colliding with obstacles, which is merely something to be avoided.

1.8 How this thesis is organized

Chapter 2 discusses how we use model minimization to describe 2-D grasping problems as finite-space POMDPs whose optimal policies can be solved off-line using approximate POMDP solvers. Chapter 3 introduces the concept of a World-Relative Trajectory (WRT) and describes how to use forward search POMDPs to solve the problem of grasping 3-D mesh objects. Chapter 4 presents experimental results for the WRT POMDPs in simulation and with an actual 7-DOF Barrett Arm and hand. Chapter 5 summarizes the contributions made in this thesis and discusses future research directions.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Grasping as a finite-space POMDP

2.1 Introduction

The challenge in modeling manipulation as a POMDP is in dealing with continuous state, action, and observation spaces. In this chapter, we use compliant motions to turn a continuous action space into a small number of discrete macro-actions. We perform state aggregation to reduce the continuous state space to a set of discrete, abstract states. Finally, we limit the observations to a small set of discrete contacts, ignoring the robot's continuous proprioception.

By building an abstraction of the underlying continuous configuration and action spaces, and ignoring proprioception, we lose the possibility of acting optimally, but gain an enormous amount in computational simplification, making it feasible to compute solutions to real problems. Concretely, we will use methods of model minimization to create an abstract model of the underlying configuration space, model the problem of choosing actions under uncertainty as a POMDP, and use an off-line policy solver to generate approximately optimal policies.

2.1.1 State and action abstraction

Robot manipulation problems are typically framed as having high-dimensional continuous configuration spaces, multidimensional continuous action spaces (positions or torques), possibly continuous time, and nearly-deterministic dynamics. However, accurately predicting the dynamics of robots contacting objects in continuous action and configuration spaces is excessively difficult. Also, if those actions are compliant and change depending on sensor values that are noisy, they lose their nearly-deterministic nature. Our approach to dealing with these problems will be to construct discrete abstractions of the robot's state and action spaces, and to make stochastic models for predicting the effects of actions.

It is possible to use a grid discretization of the continuous belief space, but the high dimensionality of that space makes it infeasible for most problems of interest. Instead, we pursue a discretization strategy that is more directly motivated by the uncertainty in the problem. When there is uncertainty with respect to the configuration of the robot or obstacles, we will generally want to execute actions that reduce uncertainty,

while making progress toward a goal. There are two ways to reduce uncertainty through action: one is to act to obtain observations that contain information about the underlying state; the other is to take actions that are "funnels," mapping large sets of possible initial states to a smaller set of resulting states.

We start by considering the MDP, defined over complete configurations of the robot and object, that underlies our problem, and construct abstract state and action spaces and an abstract state transition model on those spaces. Of course, when the robot is operating, it will not have complete information about its low-level configuration or its abstract state in that MDP. So, we will use that abstract MDP as the basis for an abstract POMDP.

We construct the abstract space for the MDP by choosing a set of abstract actions [53] and using them to induce the state space. We will work with a set of "guarded" compliant motions as our action space. A guarded motion causes the robot to move along some vector until it makes or breaks a contact or reaches the limit of the workspace. Our action set includes guarded motions through free space, as well as compliant motions, in which the robot is constrained to maintain an existing contact while moving to acquire another one. Note that these actions serve as "funnels," producing configurations with multiple contacts between the robot and an object, and generating information about the underlying state. In the current work, we allow the robot to move only one degree of freedom at a time: there are motions in two directions for each DOF, which attempt to move in the commanded direction while maintaining the existing set of contacts, if possible.

Abstraction methods for MDPs [19], derived from abstraction methods for finite-state automata, take an underlying set of states, a set of actions and a reward function, and try to construct the minimal abstract state space. This new state space is a partition of the original state space, the regions of which correspond to the abstract states. The abstract space must have the properties that, for any sequence of actions, the distribution of sequences of rewards in the abstract model is the same as it would have been in the original model, and, furthermore, that any two underlying states that are in the same abstract state have the same expected future value under the optimal policy.

So, given a commitment to guarded motions as our action set, the known deterministic continuous dynamics of the robot, and a specification of a goal (or, more generally, a reward function), we can apply model-minimization methods to determine an abstract model that is effectively equivalent to the original. We obtain a large reduction through not having to represent the free space in detail, because after the first action, the robot will always be in contact, or in a very limited subspace of the whole free space. In addition, large regions of the state space will behave equivalently under funneling actions that move until contact.

While it is moving in free space, the dynamics of the robot are well modeled as being nearly deterministic. As it begins to interact with objects, it is much more difficult to predict exactly what the effects of a commanded action will be; not only must we predict what pose the robot will end up in, but also whether the object will slide, roll, or be knocked over.

We begin by assuming an idealized deterministic dynamics model, derived from

the geometry of the robot and object, both in free space and in contact, and construct an abstract state space using those dynamics. Given that abstract state space, we will go back and compute more realistic transition probabilities among the abstract states. Finally, we will feed this resulting model into an approximate POMDP solver to derive a policy.

2.1.2 2-D box-grasping example

Throughout this chapter, we will use the problem of grasping a box in 2-D as an illustrative example. Consider a two-dimensional Cartesian robot with a parallel-jaw gripper, and a box sitting on a table. The robot has contact sensors on the tip and both sides of each finger, and can also detect when it is at the boundary of the workspace.

The action space is the set of compliant guarded moves *up, down, left, right, open, close*, and *lift*. If the robot has a contact in one direction, say down, and tries to move to the left, it does so while attempting compliantly to maintain contact with the surface beneath. The robot moves until its observed contacts change. A motion can also be terminated when the robot reaches its limits (we'll assume a rectangular workspace).

The robot's goal is to have the box between its fingers, at which point it can execute the *lift* action, which lifts the box, and terminates the trial. If the robot lifts when the box is between its fingers and the fingers are closed, it gets a reward of +15. If it lifts in any other configuration, it gets a reward of -50. Additionally, it gets a reward of -1 on each step, and a reward of -5 if it hits a workspace limit, to encourage shorter trajectories, and discourage long motions that leave the vicinity of the box. Our goal is to find a policy that maximizes the discounted sum of rewards.

The configuration space for this robot with the fingers open is two dimensional, with reachable configurations everywhere except for those in which the hand collides with the box or the table, or goes out of bounds. We can draw the configuration space of the robot with the fingers open as a 2-D plot, where a point in the plot represents the location of the center of the palm of the hand. Figure 2-1 shows the configuration space with points annotated that represent the hand being in free space, while Figure 2-2 shows points in the configuration space that represent the hand being in contact with either the box or the table. Grey areas are regions where the hand is in collision with the hand or the table, or out of the allowed workspace. The left grey protrusion from the bottom represents configurations in which the right finger of the hand collides with the box, and the right grey protrusion represents configurations in which the left finger of the hand collides with the box. Our state space, which we will call S, consists of all feasible configurations in the configuration space, including those in which the hand is in contact with the object (but not those in collision).

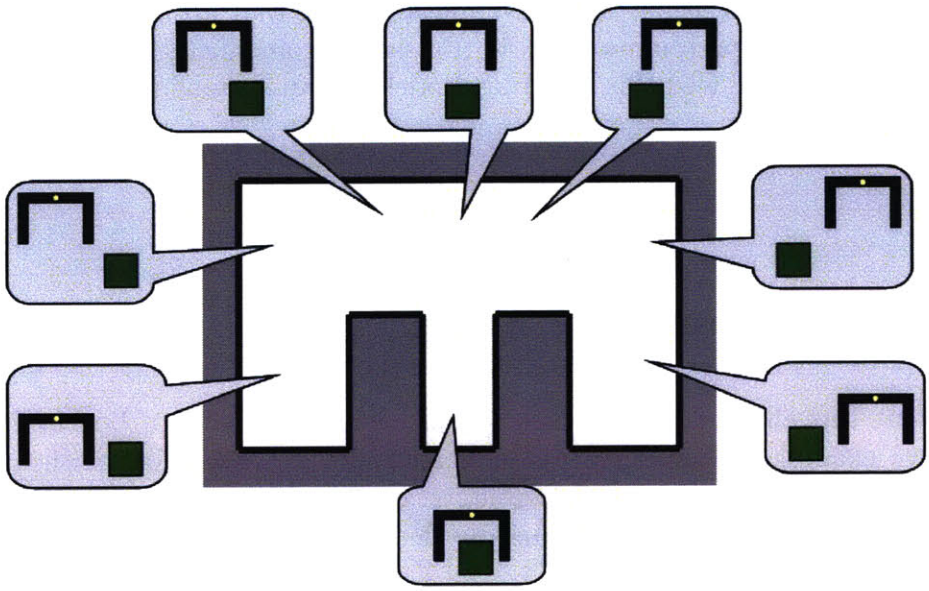

Figure 2-1: Diagram of parts of the free configuration space.

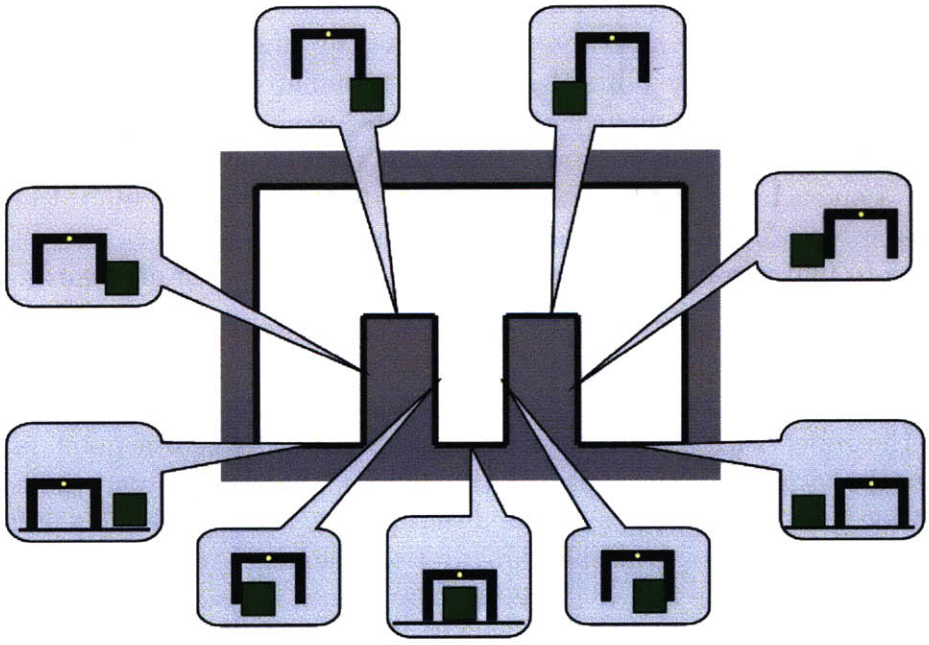

Figure 2-2: Diagram of the configuration space in which the robot is in contact with the box or the table.

2.2 Model construction

We construct the initial model by using a simple, purely geometric "simulation" of the compliant guarded moves. This geometry-only simulation is nearly deterministic, with small amounts of non-determinism due only to numerical precision. It is appropriate to think of this simulation somewhat analogously to the simulation that happens in a planner: it is a computational process, in the "head" of the robot, that will eventually lead to a plan. Note that the derived model can be re-used with different reward functions, to plan for different goals in the same environment.

In our implementation, the robot and the objects in the world are all modeled as polygons, and the simulator operates in a state space that includes the joint-space configuration of the robot and the contact state. The contact state specifies, for each vertex and surface of the robot, which (if any) vertex or surface of the world it is in contact with. All of the examples in this chapter are done in a two-dimensional world space, with robots of 2 or 3 degrees of freedom. The approach extends naturally to three-dimensional world spaces, although the number of possible contacts grows quickly. The robot is assumed to have a set of contact sensors that can give some of the information in the contact state. Our typical model will be that the robot can tell what vertices or surfaces of the robot are in contact, but not with what vertices or surfaces in the world.

2.2.1 Constructing the abstract action space

The abstract actions consist of two guarded, compliant move commands for each degree of freedom, including the gripper (one action in each direction). When there are no contacts in the starting state, the robot simply moves along the commanded degree of freedom, holding the others constant, until the observed contacts change (a new contact is made, or an existing contact is lost).

When the robot is already in contact, it is controlled by a sliding model, which is related to a damper model used by [33]. There are three cases of interest:

1. The current contact is in the opposite direction of the commanded motion. In this case, the robot simply breaks the contact and moves away through free space until it makes a new contact or reaches the limits of its workspace.

2. The current contact is in the same direction as the commanded motion. In this case, the robot cannot, and does not, move. The action is terminated if there is no motion over a short period of time.

3. The current contact has a component that is orthogonal to the commanded motion. In this case, the robot seeks to move the degree of freedom in the commanded direction (for instance, "down" or "left"), while maintaining any existing contact (for instance, a "tip" contact with the floor). It does this by making small motions in the commanded direction, then moving to regain the contact if the contact is momentarily lost. This can result in sliding along a surface, closing a parallel-jaw gripper while maintaining contact between one

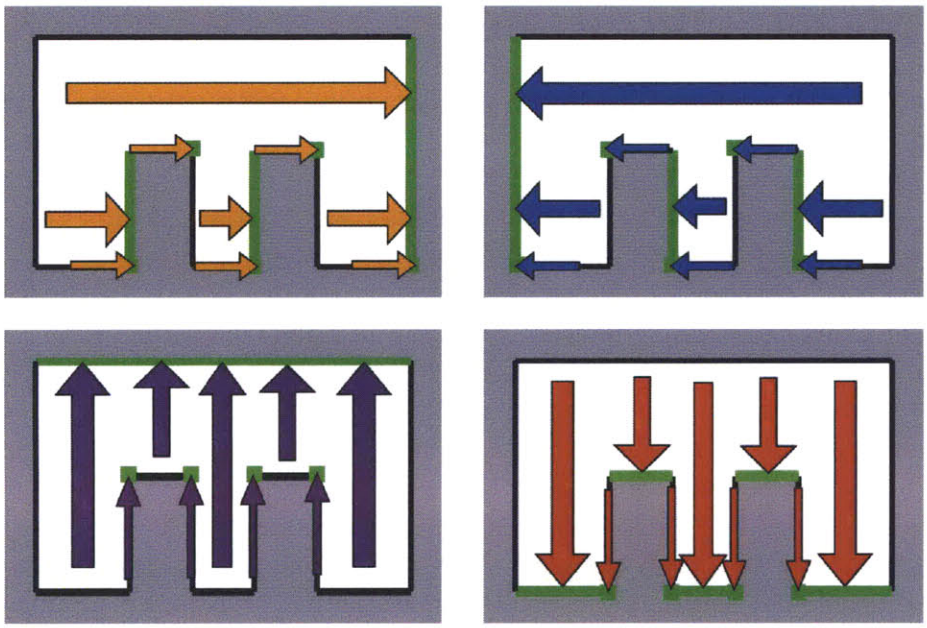

Figure 2-3: Guarded compliant actions: move until contact change.

finger and the object, or pivoting down to complete a grasp while maintaining contact of one finger on the object.

The action is terminated when the observed contacts change or when the current contact no longer has a component orthogonal to the commanded motion. Diagrams of the effects of these compliant guarded moves for our box grasping example are shown in Figure 2-3. The "funneling" effect can be seen by the way entire regions of start configurations are mapped into much smaller regions of resulting configurations. This set of abstract actions can now be used to induce a discrete abstract state space that can serve as a basis for our POMDP model.

2.2.2 Creating an abstract model

Following [19], we begin by considering how to create an abstract model of a discrete system with deterministic transitions and observations. Let $T(s, a)$ be the deterministic state transition function, specifying the state resulting from taking action a in state s; let $O(s)$ be the deterministic observation function specifying the observation resulting from being in state s; and let $R(s)$ be the deterministic reward function.

Given a low-level state space S, our goal will be to find a partition Φ, which consists of a set of non-overlapping regions P_i, such that $\bigcup_i P_i = S$. We also define ϕ to be the function that maps states to partitions, $\phi : S \mapsto P_i, P_i \in \Phi$, and so $\phi(s)$ is the region to which primitive state s is assigned. The partition Φ should also be the minimal partition that satisfies two requirements. First, that it is uniform with respect to rewards and observations, so that, for every P_i in Φ, there exists a reward r and observation o, such that for all s in P_i, $R(s) = r$ and $O(s) = o$. Second, that

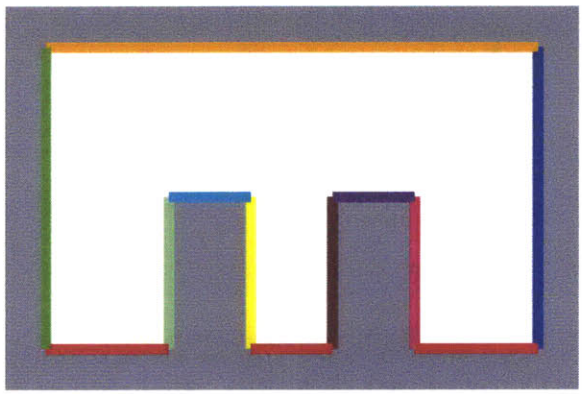

Figure 2-4: Diagram of parts of the configuration space with uniform observations. All locations with the same color have the same observation.

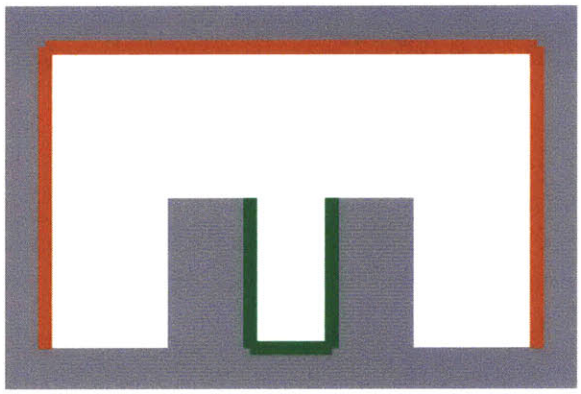

Figure 2-5: Diagram of parts of the configuration space with uniform reward. All locations with the same color have the same reward.

it is uniform with respect to *action sequences*, so that for every sequence of actions $a_1 \ldots a_n$ and every region P_i, there exists a resulting region P_j, such that for all s in P_i, the result of taking that action sequence in that state is in the same partition:

$$\phi(T(T(\ldots T(T(s, a_1), a_2), \ldots), a_n) = P_j \quad .$$

For our box grasping example, Figure 2-4 shows the parts of the configuration space that are uniform with respect to observations, and Figure 2-5 shows the parts of the configuration space that are uniform with respect to reward. Because of their ability to funnel entire blocks of states into smaller blocks of states, our actions induce a partition on the configuration space, such that configurations in each partition are uniform with respect to action sequences. This is shown in Figure 2-6.

The algorithm for finding such a minimal partition is a relatively simple process of region-splitting. We'll say that a region P_i is *deterministically uniform* with respect to action a and partition Φ if there exists a region P_j, such that for all s in P_i, $\phi(T(s, a)) = P_j$.

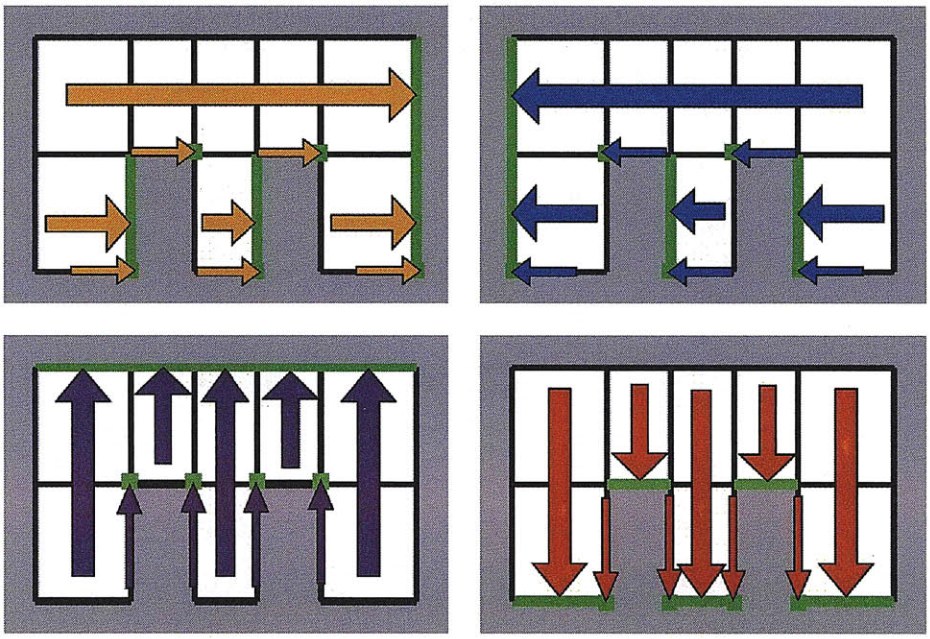

Figure 2-6: Guarded compliant actions induce a partition on the configuration space. All locations within each of the black-outlined rectangles ends up in the same set of smaller, green rectangles under the four move actions. This makes them uniform with respect to action sequences, and so they end up in the same partition.

- Let Φ be the partitioning of S such that each region in the partition is uniform with respect to reward and observation.
- While there exists a region P in Φ and an action a such that P is not deterministically uniform with respect to a and P, split P into a set of sub-regions P_i that are deterministically uniform with respect to a and Φ, and replace P with the P_i in Φ.
- Return Φ.

The resulting abstract state space is the set of regions in Φ. From this, it is straightforward to construct a POMDP with deterministic transitions and observations.

Unfortunately, even though our simple model of the robot's dynamics is nearly deterministic, our situation is more complex. The state space we are working in, at the lowest level, is continuous, so we cannot enumerate the states. It is possible, in principle, to construct a version of the splitting algorithm that operates on analytically represented subregions of the configuration space; in fact, the "preimage backchaining" method [33] is related to this algorithm. However, using compliant motions in a complex configuration space makes it very challenging to compute these regions analytically.

Instead, we will take advantage of our ability to simulate the results of the abstract actions, to build a "model" of the transition dynamics of these actions via sampling.

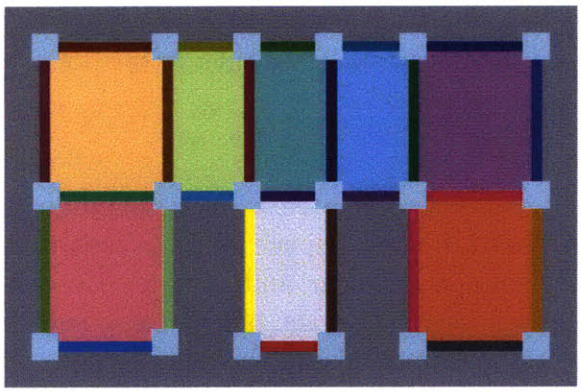

Figure 2-7: Abstract states for the box grasping problem. Note that each of the gray squares is a distinct abstract state.

We are given a set of possible starting configurations (in the case of the examples in this chapter, they were a discrete set of positions of the robot up at the top of the workspace). Based on these initial configurations, we gather a large sample of state-action-next-state ($\langle s, a, s' \rangle$) triples, by trying each possible action at each initial state, then at each of the states s' reachable from an initial state, etc.

Because of the nature of the action set, if the simulation were truly deterministic, we would expect this process to "close" in the sense that eventually no new states would be found to be reachable. In practice, due to numerical sensitivities in the simulation of the compliant motions, exact closure doesn't happen. We handle this problem by clustering reachable states together whenever they have equal contact conditions and geometric distance between robot configurations less than a fixed threshold. We draw samples until we have experience of taking each action from each cluster.

Now, each of these clusters is treated as a primitive state, and the most frequent outcome under each action is defined to be its successor. This data is now used as input to the region-splitting algorithm described above.

For our box grasping example, there are 56 resulting abstract states with the gripper fully open, shown in Figure 2-7.

2.3 Solving the POMDP

Given our abstract model, we can use an off-line optimal policy solver to find a POMDP policy. For our simple examples, the purely deterministic policy could be solved exactly. However, even for the simplest problems, as soon as we add noise (which we will discuss momentarily), it is infeasible to solve the resulting POMDPs exactly. We have used HSVI [51], a form of *point-based value iteration*, which samples belief states that have a relatively high probability of being encountered, and concentrates its representational and computational power in those parts of the belief space.

HSVI returns policies in the form of a set of α *vectors* and associated actions. The expected discounted sum of values when executing this policy from some belief state b is

$$V(b) = \max_{\alpha_i} b \cdot \alpha_i$$

and the best action is the action associated with the maximizing alpha vector. The α vectors define hyperplanes in the belief space, and the maximization over them yields a value function that is piecewise-linear and convex. By construction, each of the α-vectors is maximal over some part of the belief space; and the space is partitioned according to which α-vector is maximizing over that region.

So, to execute a policy, we apply a state estimator as described in section 1.2. The state estimator starts in some initial belief state, which is our prior probability distribution over the abstract states, and then consumes successive actions and observations, maintaining the Bayes optimal belief state. To generate an action, the current belief state is dotted with each of the α-vectors, and the action associated with the winning α-vector is executed.

2.3.1 Deterministic policy and example sequence

Figure 2-8 shows a policy graph for the POMDP policy we automatically derived for our two-fingered box grasping example, in the absence of noise. In a policy graph, the nodes are labeled with actions and the arcs with observations. In this graph, the contacts for the left finger and right finger are shown on the arcs separated by a comma, e.g. (in-tip,nil) indicates *inside* and *tip* contacts detected on the left finger and no contact detected on the right finger. Each finger has tip, inside and outside sensors.

From the policy graph, we see that the policy first asks the robot to move down; then, depending on the observation that is made, it selects a strategy. If it feels two tip contacts, (tip,tip), then there are three possible situations: the fingers straddling the box, completely to the left of the box or completely to the right. It moves to the left, and now there are three possible observations: (tip,in-tip), which means that the right finger contacted the box and so the fingers are straddling the box, (tip,tip,-X), which means the robot got to the negative-x limit of the workspace and so must have started completely to the left of the box, and (out-tip,tip), that is, the left finger touched the box on its outer sensor and so must have started completely to the right of the box. The rest of the policy proceeds in a similar fashion. This whole policy is represented internally by 385 α-vectors.

Figure 2-9 shows an example sequence in which this deterministic policy is executed. The robot always starts near the top of the workspace, and does not know where it is relative to the box, so the initial belief state is uniform over the free space configurations above the box. In this example, the true starting state of the robot is entirely to the right of the box.

In its first action, the robot moves down, and our deterministic transition model changes our belief state as shown in the second configuration space diagram on the left, which indicates that by moving down, the robot must either now be touching

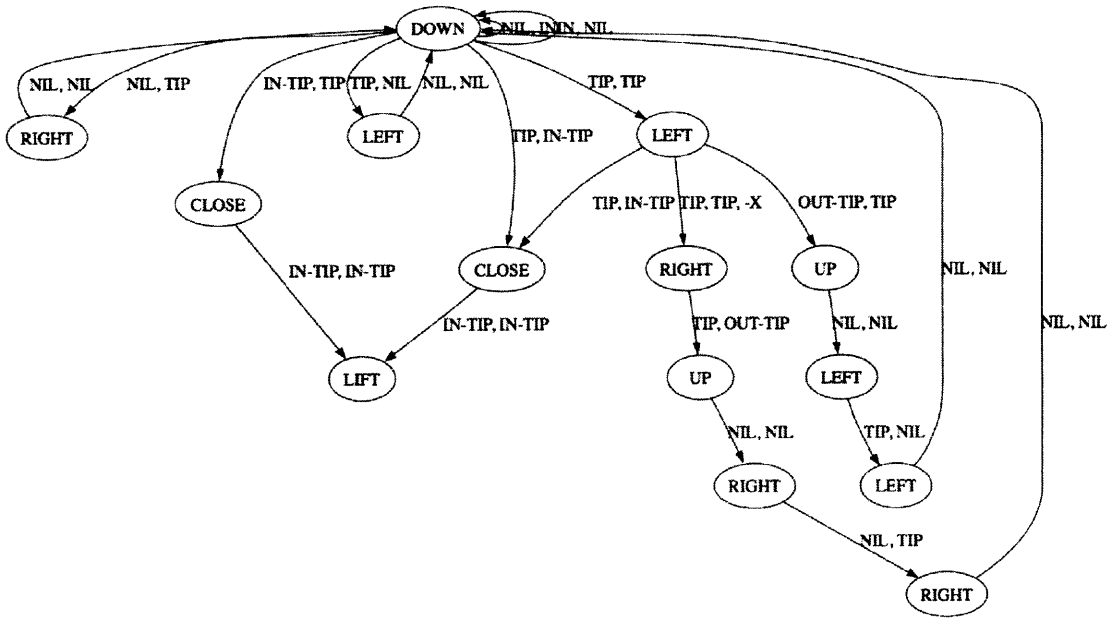

Figure 2-8: Policy graph for the deterministic POMDP for the 2-fingered grasp of a box.

the table or the top of the box.

The robot then observes (tip,tip), and our deterministic observation model changes our belief state as shown in the third diagram on the left, which indicates that by seeing contact on both fingers, the robot could not be touching the top of the box (where it would see only one finger contact), and thus must be either all the way to the left of the box, straddling the box, or all the way to the right of the box, just as in our policy graph example.

In the second action, the robot moves left, with the same possible outcomes as described in our policy graph example. It then observes (out-tip, tip), and now it knows that it is all the way to the right of the box.

At this point, having completely localized itself, and being in a completely deterministic world, actions 3-7 simply move the hand up and around the box until the fingers are straddling the box, with the belief state tracking the progress of the hand. At that point it can close the fingers, lift, and receive its reward for successfully grasping the box.

2.3.2 Adding stochasticity

The real world, of course, is not deterministic, and so we need observation and transition models that reflect the stochasticity of our actions and observations. We take a very simple approach to adding stochasticity. For observations, we assume that the contact sensors have some probability of failing to detect a contact (independent failures for each sensor), but no chance of mistakenly sensing a contact when there

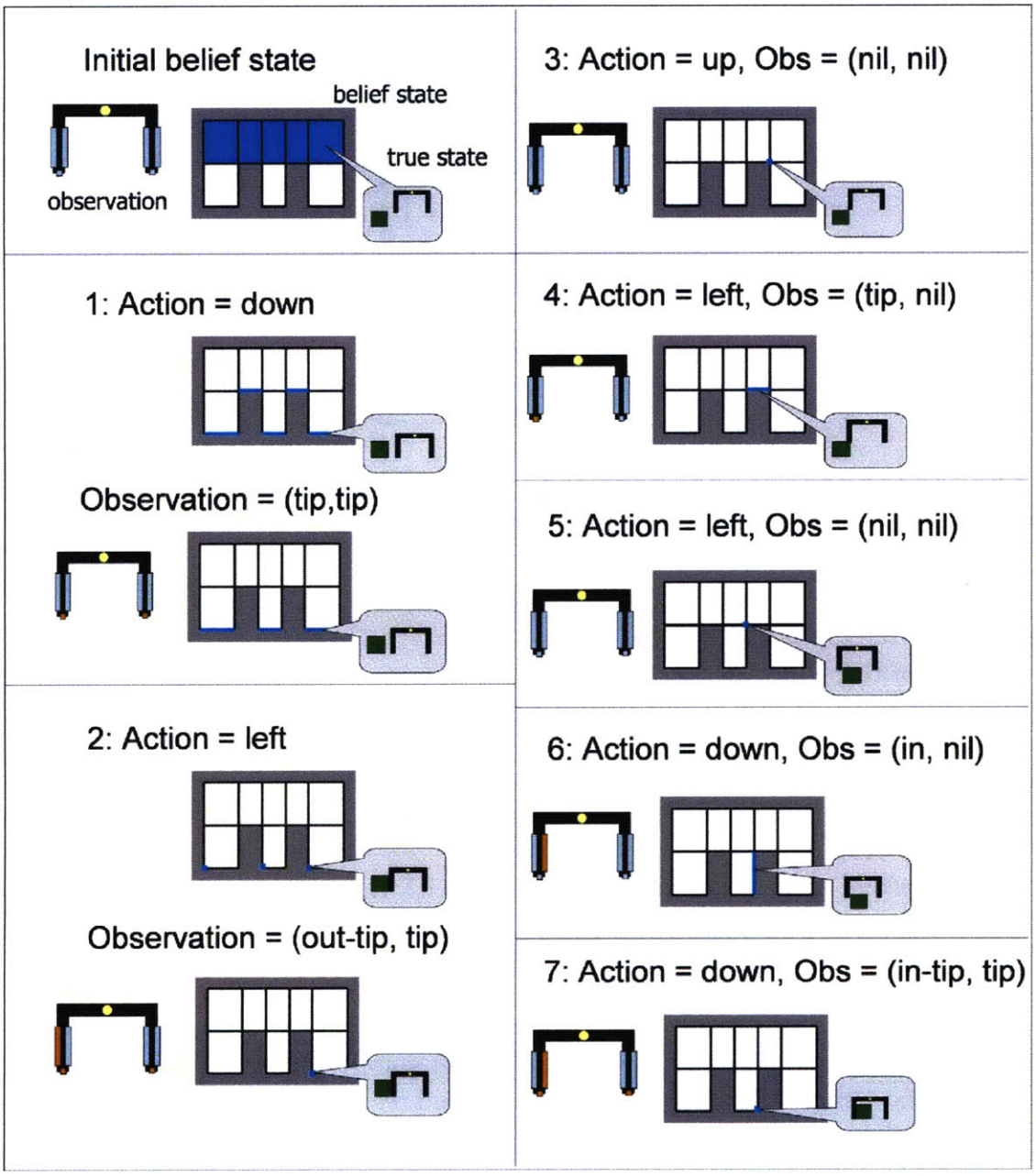

Figure 2-9: An example sequence for a deterministic two-fingered box grasping problem.

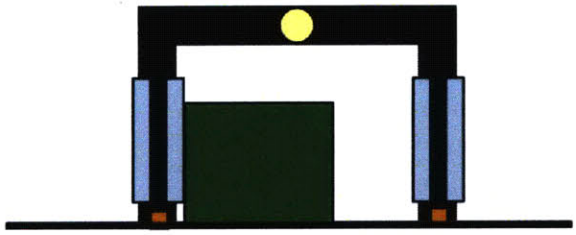

Figure 2-10: Contacts are sometimes incorrectly sensed (the finger should be seeing the contact with the side of the box).

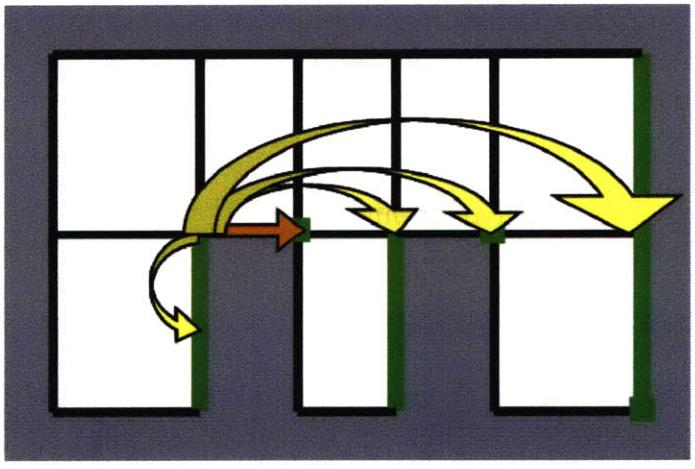

Figure 2-11: Adding stochastic transitions to nearby states.

shouldn't be one. Figure 2-10 shows an example, in which the hand should be sensing both the tip and the inside contacts of the left finger, but instead senses only the tip sensor.

For transitions, we add two forms of stochasticity. First, we reason that, in executing a particular action a from a particular abstract state P_i, it might be possible to reach, in error, any of the other states that can be reached from P_i using other actions. So we add some probability of making a transition from P_i to P_j under action a, when there exists any a' such that $T(P_i, a') = P_j$. Second, we note that there are some states, such as those involving single-point contacts, that the robot is very likely to overshoot by not noticing the relevant contact changes, which we call "unstable" states. So, for any state P_j that is unstable, and such that $T(P_i, a) = P_j$ and there exists an action a' for which $T(P_j, a') = P_k$, we add a non-zero probability of a transition from P_i to P_k under action a. In addition, if one of these resulting states P_k is unstable, we add transitions to its successors as well. An example of adding such stochastic transitions from a single state under one action (the right finger tip being at the top left corner of the box, and taking the action *right*) is shown in Figure 2-11.

This is a very simple model of the stochasticity in the system, which is certainly inaccurate. One advantage of using POMDPs for control is that they are generally quite robust with respect to variations in the transition and observation probabilities. It would also be possible to further tune the error model using a high-fidelity dynamics simulation or even data from a physical robot.

Given our newly stochastic observation and transition models, we can feed the entire abstract model into an approximate POMDP solver, as described in section 2.3, and obtain a POMDP policy that takes into account noisy observations and actions. A partial policy graph for the box-grasping POMDP with stochastic transitions is shown in Figure 2-12. In order to make the graph as small as it is in this figure, it was necessary to remove any paths with probability less than 0.01, which accounts for the nodes with no descendants. Unlike the deterministic policy, it would be difficult to imagine wanting to even attempt to write such a policy by hand.

The entire process of computing stochastic abstract models is quite efficient, even if the policies were slow[1] to derive off-line.

2.4 Simulation results

As a proof of concept, we have tested the approach described above in two planar problems: one for two-finger grasping of a box, as in our example, and one involving placing one finger on a stepped block. We derive stochastic policies from simulations on a simple planar model. We then run the policy for the stochastic model in a high-fidelity dynamics simulation and measure average total reward per episode. Note that the stochastic model and the high-fidelity model differ in some substantial details: the dimensions of the object and the geometry of the fingers are different and the actual

[1]For large problems, the POMDP approximation methods may become slow, but for all the results reported here, the POMDP solutions ran in under 10 minutes, and now there are faster solvers such as SARSOP [26].

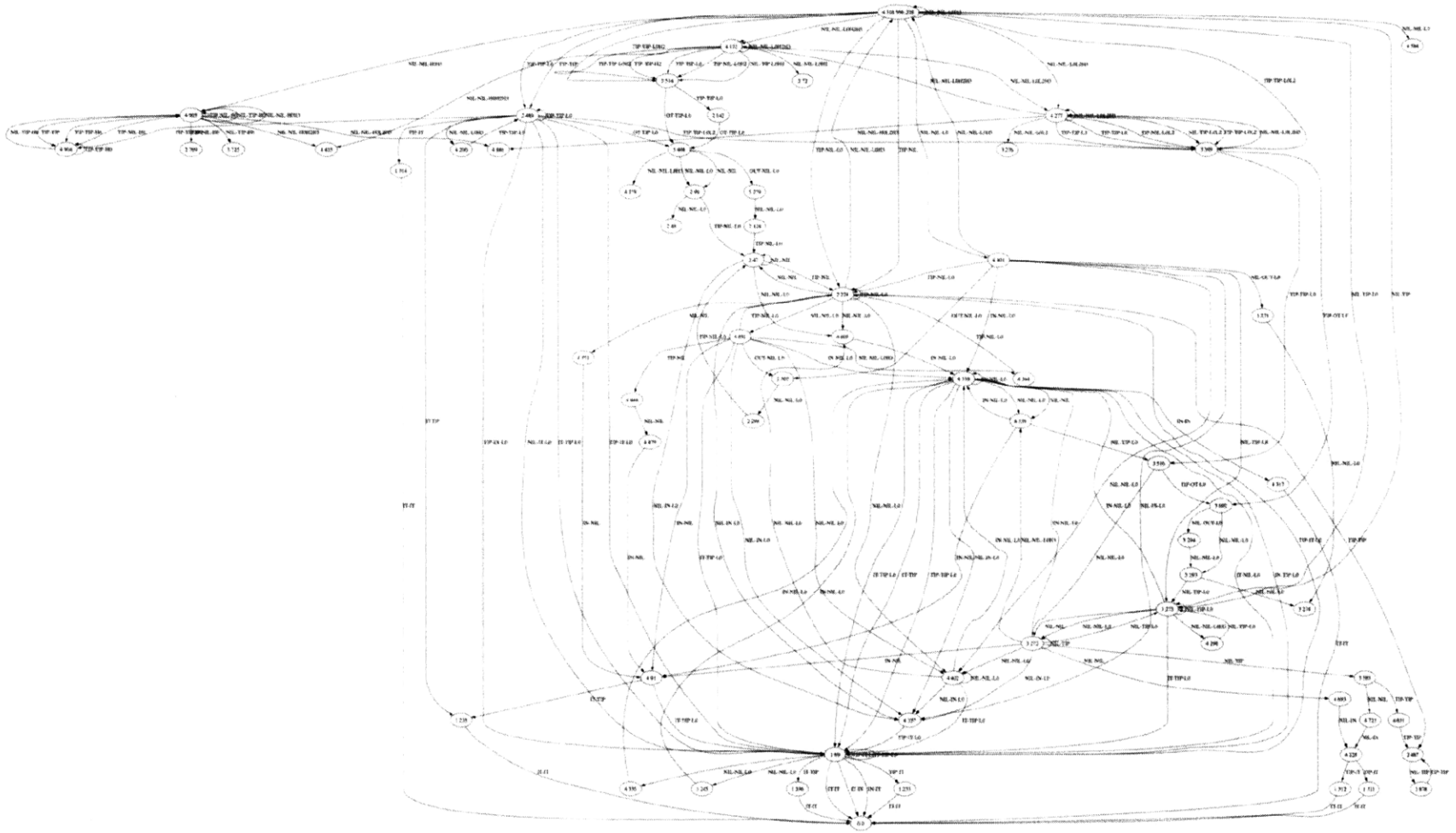

Figure 2-12: Partial policy graph for the POMDP with stochastic transitions for the 2-fingered grasp of a box.

	Results	Average Reward
POMDP policy	466/506 (92%)	-1.59
Fixed policy	154/190 (81%)	-10.63

Table 2.1: Results for placing one finger at the corner of the step of a stepped block.

sensor and detailed control behavior are different. Therefore, some of the trajectories that are most common in the high-fidelity simulation have relatively low probability in the stochastic model. These simulations gives us a measure of how much the mis-estimation of the probabilities in the stochastic model decreases performance. As a comparison, we report the results for a simple but reasonable fixed strategy, as well.

Single finger/Stepped block: In this domain, the object has somewhat more complex geometry than in our example. Instead of a simple block, the object is "stepped". The robot has only a single finger, however, and thus no *open* or *close* actions, and the *lift* action is simply a signal for success. The goal is to place the finger in the corner at the left step. Note that, since the robot is lacking position information, the goal is locally indistinguishable to the sensors from the corner where the block rests on the table. The rewards are similar to our example problem (+15 for reaching the goal, -50 for lifting in the wrong state, -1 for each motion, -5 for being in the states at the limits of the designated problem workspace). The abstract state space for this problem has 40 states.

A trajectory derived by following the stochastic policy for this problem is shown in Figure 2-13. Results are shown in Table 2.4 comparing the POMDP policy formulated as described above to a fixed policy that simply moves the hand in a fixed pattern of *left*, *down*, *right*, *right*, *up*, *right*, *right*, *right* (*LDRRURRR*). The POMDP policy is considerably more robust than the fixed strategy.

To help gain intuition about the kinds of strategies being developed, we can examine the solution for the deterministic version of this problem as well (solutions for the noisy versions of these problems are very difficult to understand intuitively; see figure 2-15 if you can). Figure 2-14 shows the deterministic policy for placing the finger at the left step of the stepped block. It first asks the robot to move down; then, depending on the observation that is made, it selects a strategy. If it feels a tip contact, then it moves to the left, and now there are three possible observations: *none*, which means it was on top of the left step or the top of the block, and has now lost contact, *tip(-x)*, which means that it is at the negative-x limit of the table, and *left&tip*, which means that it's feeling contact on the outer (left) part of the finger and the tip, so it is either on top of the right step of the block, or the right part of the table. The simplest situation is when it feels *none*: now it goes right and regains the tip contact, and moves right again. If this move results in *none* as an observation, then we know we're on top of the block (just off the right corner), and have to move back and down to the left-hand step. On the other hand, if we get *right&tip* as the observation, we know the robot is in the right place, and we can command a "lift" action.

Two-fingered grasping: Our second domain is the one we have been using as an example all along. An example trajectory derived by following the stochastic policy

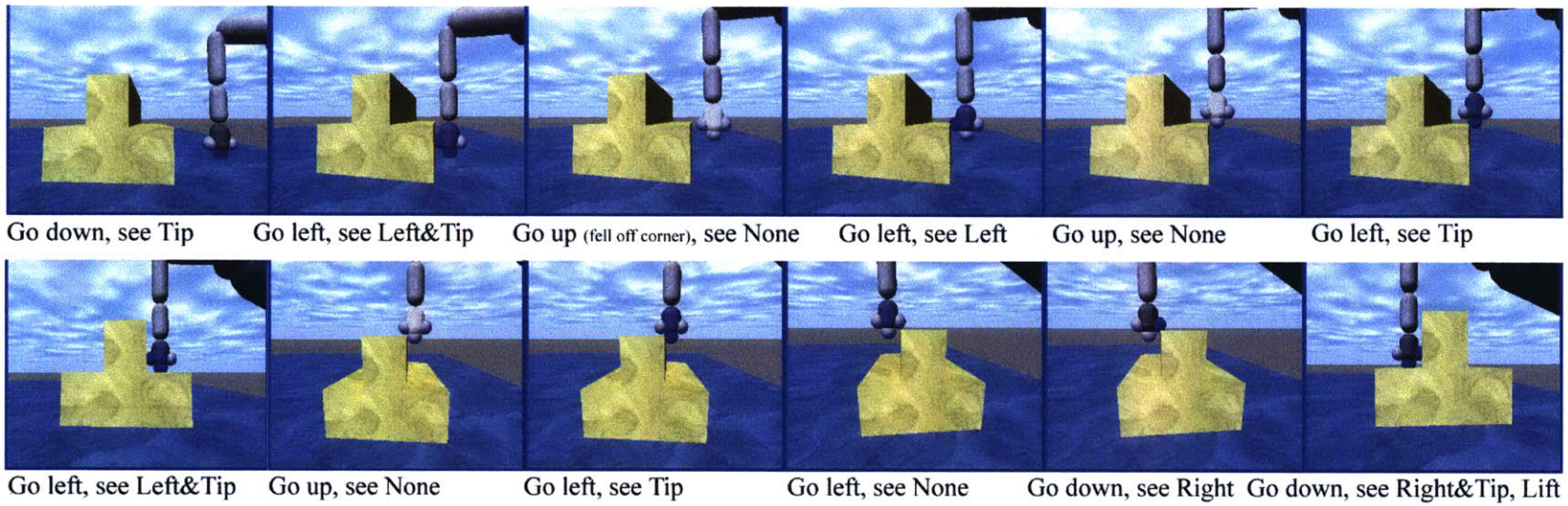

Figure 2-13: Sample run of one-finger policy on stochastic stepped block model.

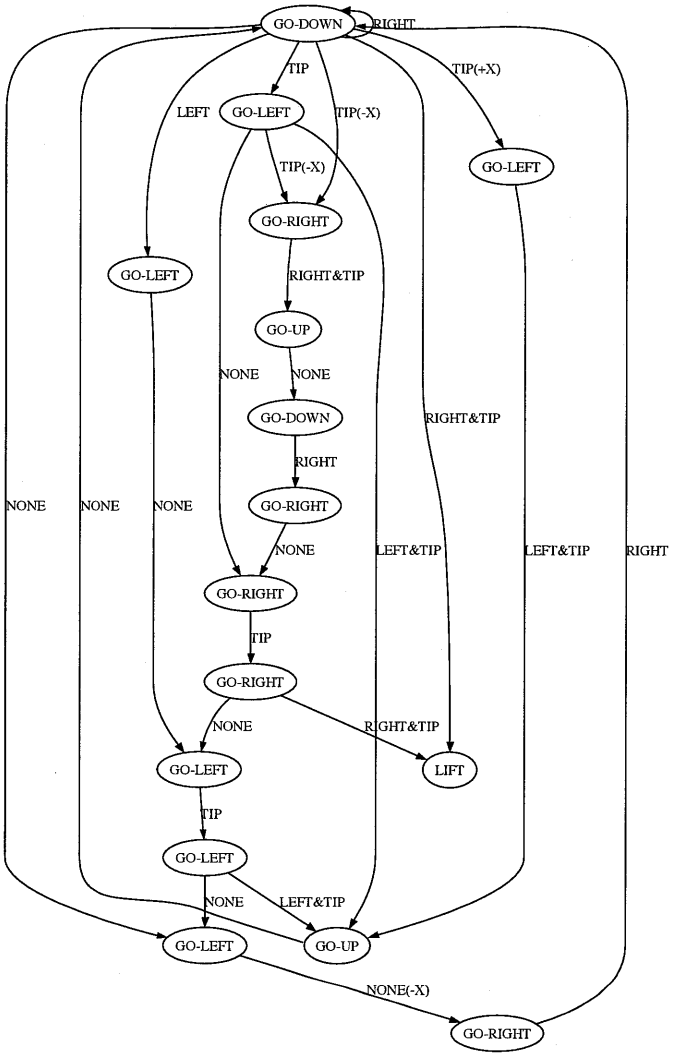

Figure 2-14: One-finger policy for deterministic stepped block model.

	Results	Average Reward
POMDP policy	115/115 (100%)	4.0
Fixed policy	86/113 (76%)	-17.24

Table 2.2: Results for two-fingered grasping of a block.

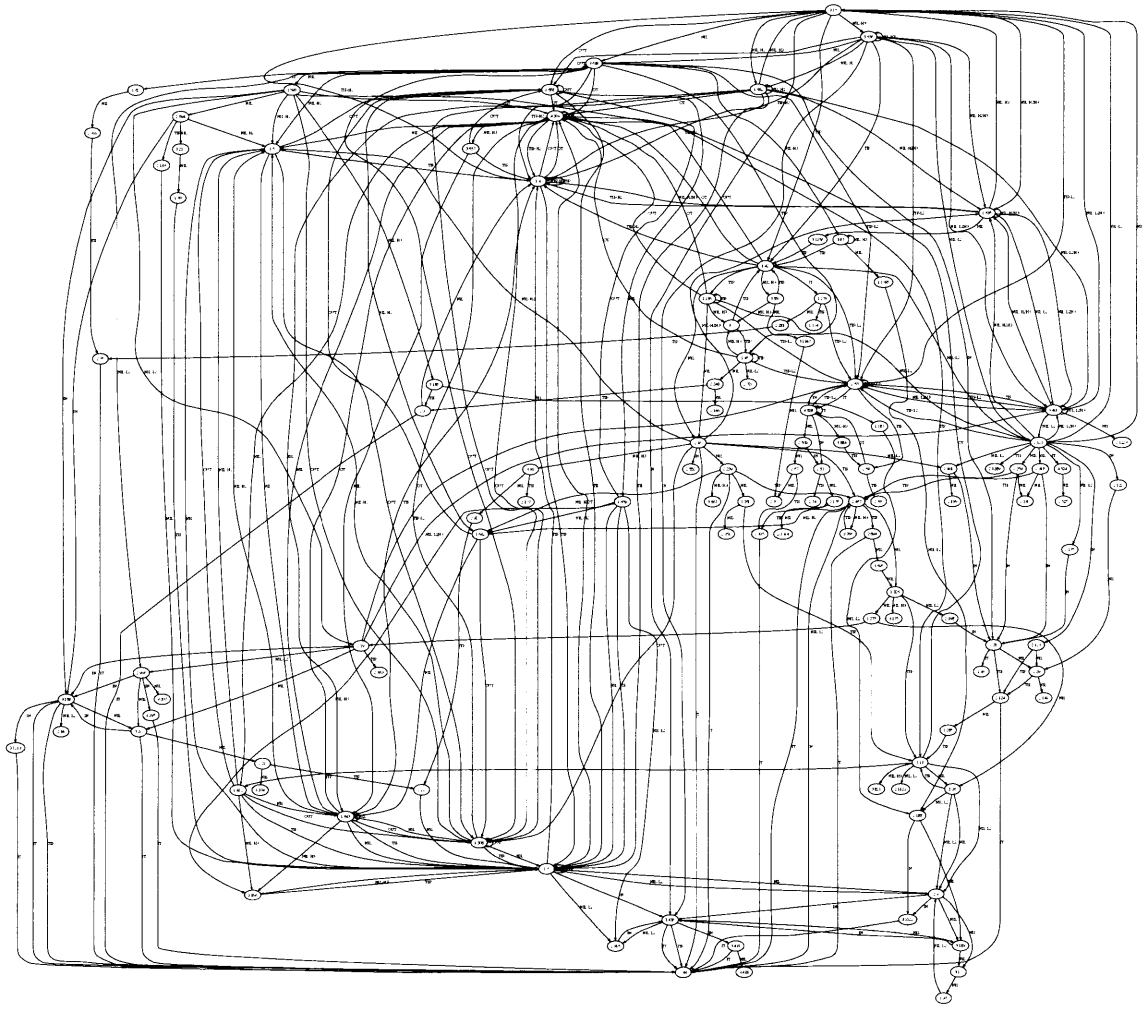

Figure 2-15: One-finger policy for noisy stepped block model.

for this problem is shown in Figure 2-16. Table 2.4 compares the results of this policy, found by solving the POMDP formulated as described above (and encoded in 1196 α-vectors), to a fixed policy that simply moves the hand in a fixed pattern of *LDRRURRDDDG*. The fixed policy performs significantly worse than the POMDP policy.

In our simulated block-grasping experiments, we varied the shape of the block a fair amount, as shown in Figure 2-17. As long as the block did not grow so large as to not fit within the hand, or so tall as to not allow the fingertips to touch the table while straddling the block, or so slanted that the surface-following actions failed, the abstract states and transitions remained the same, and thus so did the policy. (The base of the slanted block is cut off because having the slant continue all the way to the table can cause the finger's inside or outside sensor to miss seeing the side of the block when going sideways toward the block, and thus the sideways action continues on up the side of the block instead of stopping. A more shallow slant does not cause this problem.) Using the same policy on all four blocks shown in the figure allowed the robot to grasp with the same behavior and success rate. This shows that using compliant actions that look only at contact changes and not at the distance traveled can make a policy robust to a fair amount of shape uncertainty.

2.5 Experiments With a Physical Robot

We have experimented with some of these policies on our robot platform, which is a 7-dof Barrett arm with a 3-fingered, 4-dof Barrett hand. On the Barrett hand, two of the three fingers can spread together to perform pinch, tripod, or even hook grasps. When the two outer fingers are in their maximally spread position, all three fingers are together, and thus the hand can be viewed as a one-fingered robot. This configuration is shown in Figure 2-18. If the three fingers together are placed at an appropriate location on either of the two objects discussed previously, and the depth and width of the object are not too large, the outer two fingers can be swung around to perform a pinch grasp. Thus, in most of our initial experiments, we use the one-finger policies to position the hand for grasping.

2.5.1 Sensors

Fingers outfitted with sensors have three pressure sensors, inside, outside, and tip, as shown in Figure 2-19. These sensors, of our own design, use force-sensing resistors (FSRs) to provide information about both the force and location of contact. More information about the sensor design is given in Appendix A. While the finite-space POMDP policies in this chapter only make use of contact/no-contact information, the location information could be used in more complicated POMDP models, such as the ones we use in Chapter 3.

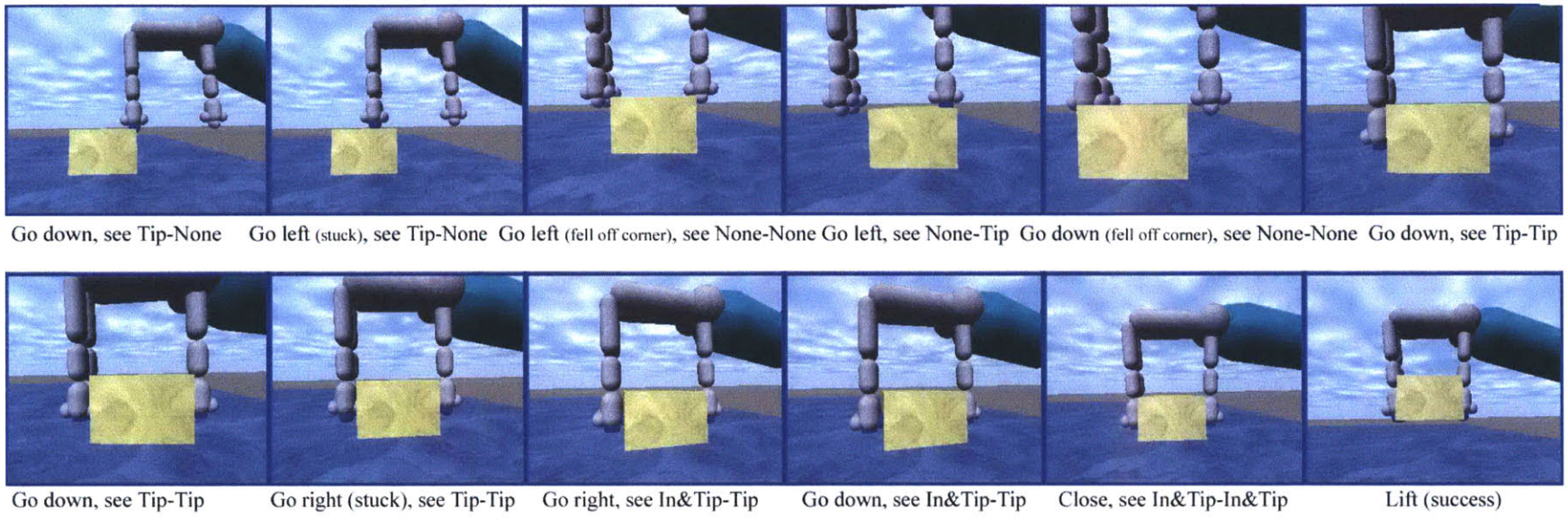

Figure 2-16: Sample run of two-fingered box-grasping policy in high-fidelity simulation.

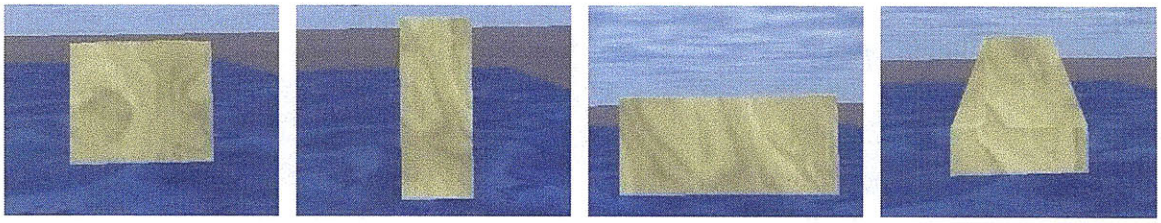

Figure 2-17: Blocks with significant shape variation that still have the same abstract states and transitions, and thus the same policy.

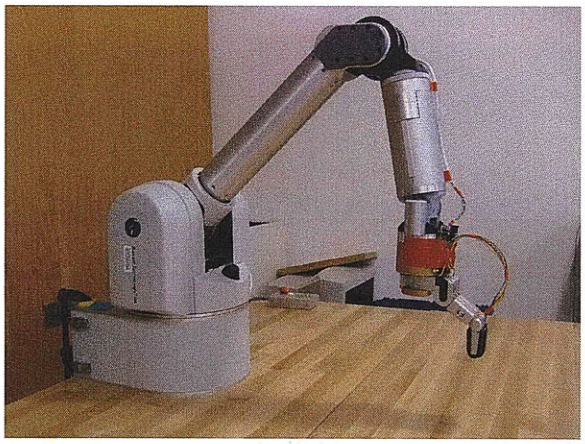

Figure 2-18: Barrett arm and hand in one-fingered configuration.

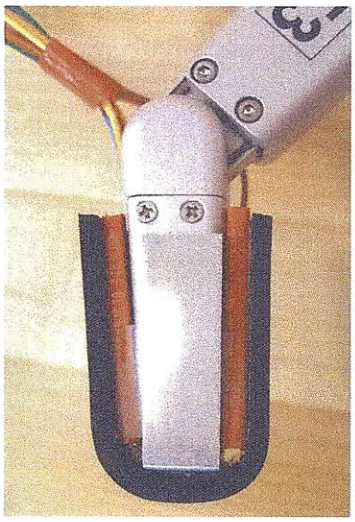

Figure 2-19: Finger with sensors.

2.5.2 Single finger results

Stepped block results: In our first set of experiments, we ran the one-fingered, stepped-block POMDP policy described above. This policy attempts to place the Barrett hand in its one-fingered configuration at the corner of the right step of a stepped block, a position from which it can grasp the top block. To compare with the POMDP results, we used a closed-loop fixed policy that goes to the right workspace boundary, then goes down to touch the table, then keeps going left until the inside sensor is triggered (placing the finger at the bottom right corner), keeps going up until the inside sensor is gone (which should leave the finger just above the right step), then alternates going left and down until the finger sees both the inside and tip sensors (at which point the finger should be at the goal location at the corner of the right step). This closed-loop fixed policy was carefully hand-designed to succeed as often as possible, and thus succeeded 49 out of 50 times, for an average reward (+15 for success, -50 for failure, -1 for each step, -5 for hitting a boundary) of 2.1. However, the one failure was not the fault of the policy–a spurious In-Tip was seen on the way to the corner, making the run indistinguishable in terms of sensor observations from a successful one; had it been a success, the reward would have been 3.4. A POMDP made to ignore the boundary cost takes nearly the same trajectory, with an extra two steps to verify that it has gone over the corner step successfully. Although it succeeded in all 50 of 50 runs, the extra steps lower the average reward to 1.6. In this particular case, the closed-loop policy nearly always succeeds because actions that move up the side of the block without backtracking have a very low rate of failure; if that were the best trajectory to take in all cases, a POMDP policy would be unnecessary. A POMDP that takes into account the boundary cost and thus goes down first to avoid hitting the right boundary, on the other hand, has to deal with an extremely high rate of failure. It is a good indicator of the robustness of POMDP policies that running such a policy succeeded in 10 of 10 runs with an average reward of 4.0, beating the hand-designed fixed policy, even in the face of a number of spurious contacts and early-aborted actions.

Since the actions had been tuned to that particular stepped block and thus had very few failures on the closed-loop fixed policy, we tried a smaller stepped block with the same action parameters to see how the policies would compare. For the smaller stepped block, moving up the side of the bottom block sometimes fails by missing the middle step. This is because while following the side, the controller tries to keep a constant pressure on the object while sliding, and while adjusting the depth, it is allowed to miss contacts for a tunable number of steps before regaining contact. This results in curving around corners before the controller decides that the contact is definitely lost and backtracks to regain contact before ending the action. On the larger block, there is room to spare to figure out that the bottom side is lost, but on the smaller stepped block, the finger can encounter the side of the top block while trying to regain contact and thus miss the middle ledge entirely. Because of this, the closed-loop fixed policy failed in 2 of 5 runs on the smaller stepped block. The same POMDP as before (that takes into account the boundary cost), on the other hand, still succeeded in 5 of 5 runs. This includes one run that goes all the way to the

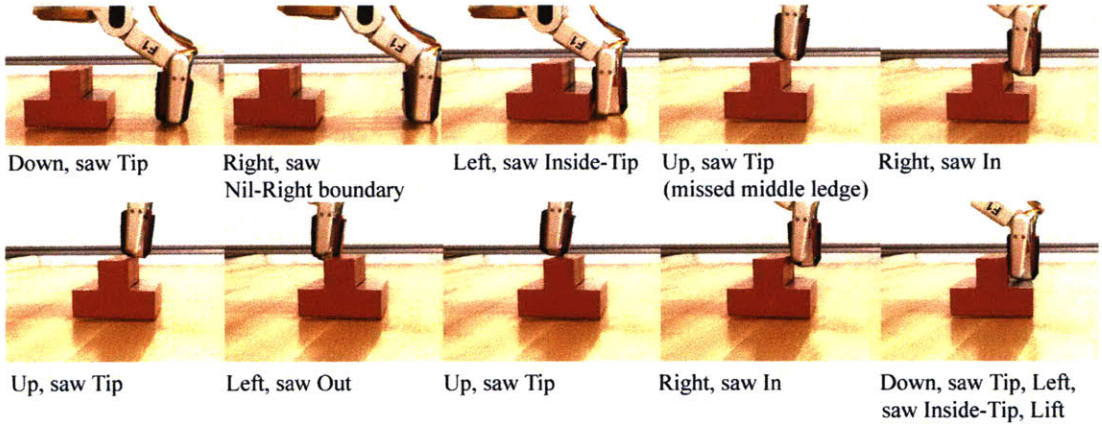

Figure 2-20: Sample run of one finger on a small stepped block dealing successfully with accidentally skipping the middle step.

right boundary before returning, missing the middle step on the way up the side of the stepped block, checking to make sure it has gone over the corner, then moving to what it expects will be the goal. At that point it unexpectedly sees an outside contact, figures out that it has missed the middle step, and recovers successfully. Snapshots from this run are shown in Figure 2-20. Because the POMDP is built to deal with unexpected transitions up to two states away in each move, it can recover from such situations with a robustness that would be difficult to achieve with hand-written closed-loop policies.

2.5.3 Two finger results

We have also run experiments on a two-fingered hand; as in simulation, our experiments involve trying to grasp a simple box. A fixed closed-loop policy (right, down, left until Out is seen on the left finger, up until Out is gone, left until In is seen on the left finger, down until Tip is seen on either finger, then grasp and lift) for this case has to do many more steps than a POMDP can get away with, but even so, it succeeded in 5 of 5 runs. The POMDP policy (which starts by going down instead) for this situation also succeeded in 5 of 5 runs, although few enough errors in the actions were seen that a POMDP policy is probably unnecessary for this situation; again, the box size is ideal for the current actions, and more errors might be seen on differently-sized boxes.

Videos of these experiments can be seen at
http://people.csail.mit.edu/abstractstatepomdps.

2.6 Chapter summary

In this chapter, we presented a framework for using finite-space POMDPs to represent grasping problems. We examined the domain of 2-D grasps of rectilinear objects,

and showed how using compliant guarded moves ("move until contact") allows us to use model-minimization methods to create a compact, abstract model. In our abstract model, the abstract states are maximal equivalence classes of configurations that have the same reward and observation, and that make the same transitions to other abstract states. For certain simple problems such as grasping a box or a stepped block in 2-D, the abstract model is small enough to solve using off-line, optimal policy solvers. We tested the resulting policies both in simulation and on a real robot, and demonstrated their robustness to large amounts of noise and even modeling error.

Trying to solve off-line for an optimal policy requires us to enumerate small, discrete sets of all the possible actions we want to use and observations we might encounter. For the simple problems described in this chapter, we were able to generate state, action, and observation spaces small enough to solve for an approximately optimal policy off-line. However, it is difficult to extend these methods beyond the limited domains presented. Even adding hand rotation, or increasing the number of facets in the objects, causes the number of abstract states to explode.

Also, by limiting our observation space to just looking at noisy fingertip contacts and out-of-bounds conditions, we are throwing away useful information that should, intuitively, make our lives easier. Our robot has reasonably accurate proprioception, and combined with contact sensing, the path the robot traces out as it moves over the object could have disambiguated the hand's location relative to the object entirely. However, trying to use this information would cause the number of possible observations we might observe to explode.

Furthermore, using an offline policy solver forces us to choose a set of actions that are independent of the belief state. Even if we are fairly certain the object is in a particular location, we cannot have a single action of the form "move to the most likely position of the object and close your fingers"; instead, we would need to have a separate action for each possible location of the object, which would cause our action branching factor to explode. Any of these changes would make solving for the optimal policy off-line intractable.

Because of these limitations, the domains in which the methods outlined in this chapter are useful are fairly limited. Even in the domains we presented, the limitations on the size of the observation space, and our resulting decision to discard a great deal of available information, make our policies rather sub-optimal. In cases where the observation space is inherently limited and can be expressed as a reasonably small, discrete set of observations, and where the problem can be reduced to a reasonably small abstract model, solving the abstract-state POMDP off-line can be an excellent method for generating highly robust policies. In problems where this is not the case, methods that can make use of more of the available sensor information, or that do not require the number of underlying states to be so small, are likely to be more useful.

In the next chapter, we will show how using compliant macro-actions that are parameterized by the most likely state, and selecting actions on-line using POMDP forward search instead of doing off-line policy generation, allows us to make use of richer sensor information and to handle more complex manipulation problems than the ones presented in this chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

World-Relative Trajectory (WRT) POMDPs

3.1 Introduction

One classic approach to combining motion and sensing to carry out tasks in the presence of uncertainty is the "most-likely state" approach. The idea is to maintain a belief state, and at each time step, choose the most likely world state, plan a motion to achieve the goal in that state, execute the motion, use any sensory information obtained during execution to update the belief and repeat. This strategy is used, for instance, in [40] to do robot navigation in partially observable environments. There are a number of drawbacks to this approach: It will not plan actions to gain information, only to achieve the goal; it requires on-line motion planning to achieve the goal; and, importantly, it requires us to know what observations are likely to result from such motions in all of the possible world states. In this chapter, we pursue a variant of this "most-likely state" approach that addresses these drawbacks: it plans explicit information-gathering actions when necessary, and does time-consuming motion planning and geometric simulation, including the construction of an observation model for belief-state update, off-line.

In the approach presented in this chapter, we continue to use POMDPs to model our problem. However, we will present a very different set of choices for our POMDP state, action, and observation spaces. While these choices make it impossible to solve for the optimal policy off-line, we can instead select actions at each time step by performing a forward search through the POMDP search tree on-line. Doing so allows us to use actions that depend on the belief state, such as "move to this position relative to the most likely state", because while we might have a continuous and high-dimensional set of possible robot motions, we can choose to search over only those motions that we think are most likely to be useful given what we know about where the object is. It also allows us to use more informative observations from a continuous and high-dimensional observation space, because we only need to consider those observations that we might plausibly obtain when starting from the current belief state.

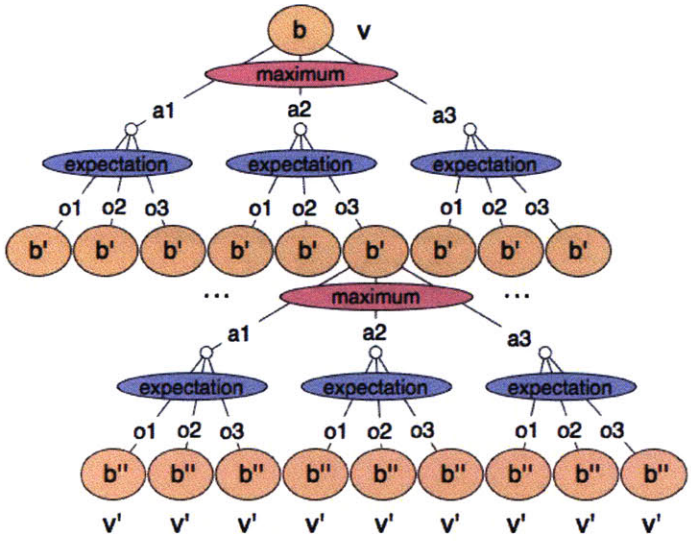

Figure 3-1: Forward search on the POMDP tree.

However, the challenges to using POMDPs for manipulation that we discussed in section 1.5 still need to be addressed even when using forward search: the state space must be discrete and small, the action branching factor and the outcome/observation branching factor must be low, the horizon must be short, and we still have to deal with the observation and transition models being computationally complex.

3.2 POMDP forward search

When using POMDPs, actions are chosen based on the current belief state, which is a probability distribution over the underlying state space. We need to be able to select an action for any belief state we might find ourselves in. To do so, we can use forward search from the current belief state as a method of selecting actions on-line.

At each time step during execution, we use a state estimator (which is simply a discrete Bayesian filter) to track the current belief state given the previous belief state, the action taken, and the observation received. We then would like to pick an action that has the highest expected future reward.

To estimate the expected future rewards for each action we construct a forward search tree like the one shown in Figure 3-1. At the top of the tree is our current belief state, b. From there, we branch on all of the possible actions. For each action, we expect that we might see a number of possible observation outcomes, so we enumerate and branch on possible observations. Each observation branch is associated with a new belief state, which is the result of doing a belief update on the parent belief using that branch's action and observation. Each new resulting belief state is akin to the root belief state; we can branch on all possible actions, to consider what would happen if we took a second action, and branch further on the observations we might

obtain after those second actions, updating the belief state again.

Repeated branching on actions and then observations continues until we reach a desired horizon H. In Figure 3-1, H is only 2, with all but one of the second-level node expansions omitted due to space constraints. When we reach the desired horizon, the leaf nodes of the tree are evaluated using a static evaluation function, which assigns a value to the resulting belief state. We can then use the tree to calculate the value of taking each action at the root node. For each action node at level H, we take an expectation over the values assigned to its observation nodes and then subtract a fixed cost for taking an action. Once the action nodes at level H have values, we can select the action with the maximum value. We continue taking maximums over actions and expectations over observations all the way up the tree to the root, which can now choose the next action to execute.

3.3 Receding horizon planning

Also important to note is that we do not search all the way down in the POMDP forward search tree to determine a policy that covers all the reachable belief states. We plan using a receding horizon, which means that at each time step, we make a plan that takes into account only the next H steps. We then take a single action, make an observation, and plan again using what we now know about the world. This approach requires a reasonably good static evaluation function for belief states, so that even with limited lookahead we can choose actions intelligently. At the same time, it allows the POMDP search to use a limited approximation of the full search tree, since we only have to pick a reasonable next action, knowing we will have a chance to plan again after the next observation.

Even when using POMDP forward search with a receding horizon, there are still a number of challenges involved in making action selection tractable. The belief state must be compact to represent and reasonably efficient to update; the action and observation branching factors must both be quite low; the horizon must be short; and we need to be able to use the observation and transition models for belief update quickly. We will now discuss how we fill in the choices of state space, actions, and observations, and how we compute the observation and transition models.

3.4 The state space and belief space

To make the state space manageable, we partition it into observable and uncertain components. We assume that the robot's position is approximately observable based on proprioception, and thus do not need to include it in the uncertain part of our belief state. A POMDP with fully and partially observable components separated in this fashion is called a MOMDP, which stands for mixed observability MDP [41].

Let Φ be the set of possible robot poses, in absolute joint coordinates, and let $\mathcal{W}$ be the space of possible configurations of the world. In the simplest case, $w \in \mathcal{W}$ will be the pose of a single object of known shape, supported by a table, and specified

by (x, y, θ) coordinates. A *belief state* of the system is a probability distribution over $\mathcal{W}$ representing the system's state of information about the world it is interacting with, together with a single element ϕ of Φ, representing the known robot pose. We represent belief states using a set of sampled points in $\mathcal{W}$ (spaced regularly on a grid) together with weights that are proportional to the probability density at those grid points.

3.4.1 Estimating the most likely state

We often wish to estimate the most likely state (MLS) of the object pose from our belief state grid. If we only use the pose of the grid point with the highest probability, we are limited to the resolution of the grid. If we could use a grid with arbitrarily high resolution, this would not be an issue; however, due to computational limitations, it is often the case that the grid is coarser than we would like our MLS estimate to be. However, under the assumption that the underlying continuous probability function is locally smooth, we can use a smoothing function over the grid probabilities to estimate the MLS of the underlying continuous function. We smooth over a small section of the grid around the grid point with the highest probability (in our implementation, the section includes points up to 2 grid cells away).

Figure 3-2 shows a few examples of representing an underlying probability function with grid point samples, and then estimating the MLS through the resulting grid points. In this figure, the grid approximation is shown for for just one dimension (x). The underlying probability distribution that we would like to represent with our grid points is shown with the dotted blue curve, and the true MLS is shown by the dashed, cyan, vertical line. We sample the underlying distribution at the blue and red circle-points, and use those values as our tracked belief state. If we use just the most likely grid point, we will often be as much as half the grid resolution away from the true MLS, as in the case of the Gaussians in c) and f), where the peak is exactly between two grid points. By using a smoothing function, we can often estimate the MLS more accurately than the grid resolution. The red circle-points are the five points (the most likely grid point and the two surrounding on each side) that go into estimating the MLS in our implementation.

We can get a fast estimate of the MLS just by taking the weighted mean of these five points. The result of doing so in the example distributions in Figure 3-2 is shown by the solid blue vertical line. For highly peaked distributions, this works quite well, as with the Gaussian functions shown in panels a)-c). However, for less-peaked distributions, in which the probabilities over the entire grid section are comparable, the weighted mean is fine if the peak happens to lie on or very near the most likely grid point, as in section d), but it is biased towards the edge of the grid section, as in sections e) and f). If the underlying function is bimodal, as in section f), using the weighted mean can be more disastrous than just using the most likely grid point. In this example, the weighted mean is almost an entire grid point away from the true MLS.

For improved accuracy, we use a smoothing function that sums up Gaussians centered at each grid point, weighted by the grid point probability, and with a variance of

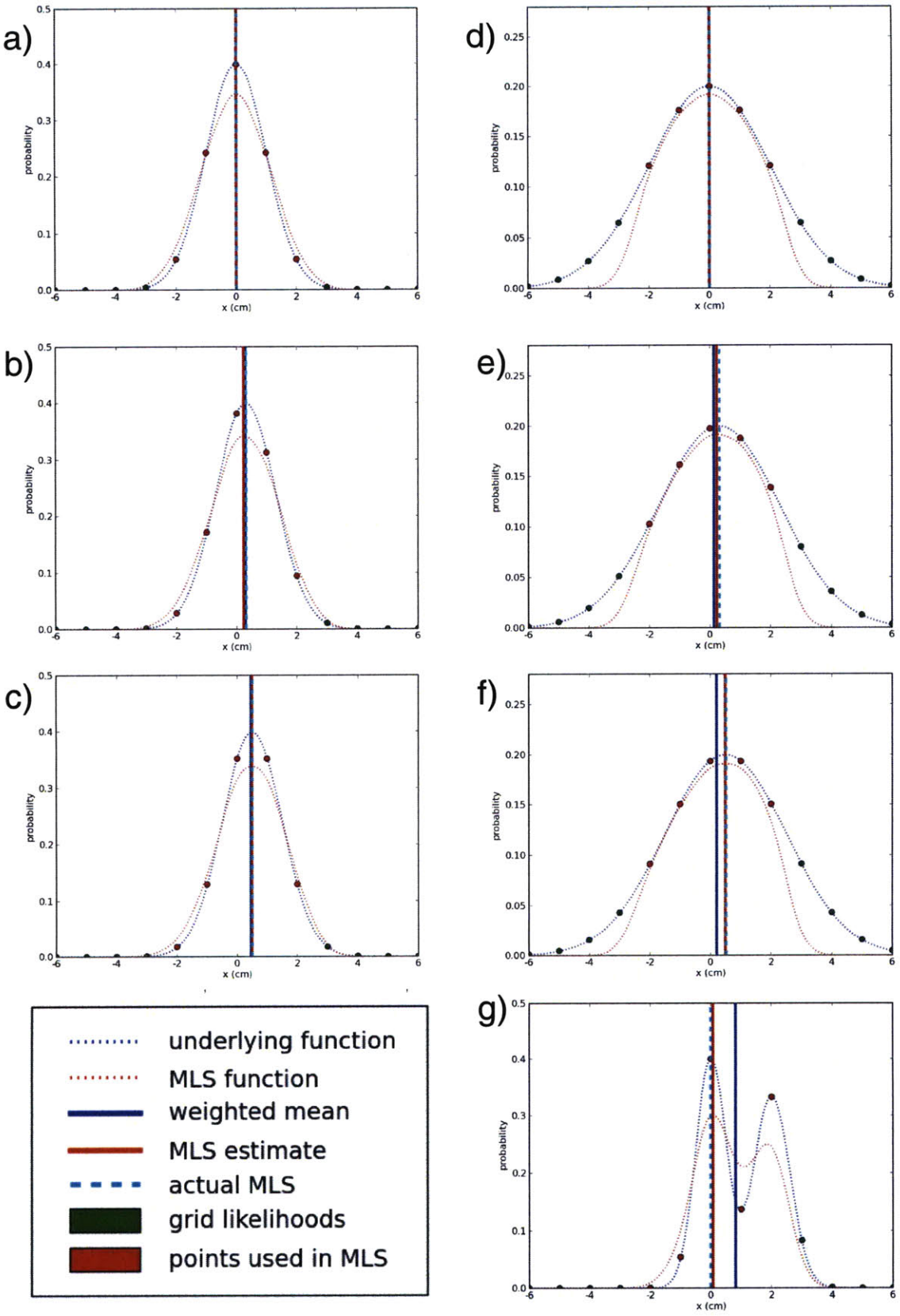

Figure 3-2: Graphs showing the accuracy of two functions estimating the most likely state of a continuous function approximated by a grid of sampled probability values.

0.6 times the grid spacing (the variance is chosen empirically). We then use numerical optimization (scipy.optimize.fmin) to find the peak of the smoothing function. The smoothing function for each example in Figure 3-2 is shown by the dotted red curve, and the peak of the smoothing function, which is our improved estimate of the MLS, is shown by the solid red line. In each of the examples, this MLS estimate is on top of or very near the true MLS.

3.4.2 Belief-state goal condition

Having described the state space of the system, we need to be able to articulate a goal condition. Most straightforwardly, we might imagine the goal to be some predicate $G(\phi, w)$, specifying a desired relation between the robot and the objects in the world (such as a particular range of grasp locations, or desired contacts between parts of the hand and parts of the object, or a combination of the two). The range of grasp locations can even include non-contiguous regions, if there are multiple locations on the object we might wish to grasp.

When desired contacts are part of the goal condition and no contacts are sensed near the desired locations on the hand, or contacts are sensed but their relative positions are out of the desired ranges, we can tell immediately that the goal condition is not satisfied. However, when the sensed hand contacts could plausibly be within the goal condition, there is typically ambiguity in terms of where the contacts are on the object.

Because of such ambiguities, having a goal condition on states of the world is not directly useful: the system will be unable to determine, with certainty, whether it actually holds in the world. So, we must formulate goal conditions on belief states, instead. We can construct a goal condition, $G_\delta(\phi, b)$ on belief states by requiring that the system believe, with confidence δ, that the goal condition holds; that is, that

$$\sum_w b(w) I[G(\phi, w)] > 1 - \delta \ ,$$

where b is a belief state, $b(w)$ is the probability that b assigns to the discrete world state w, and I is an indicator function with value 1 if its argument is true and 0 otherwise. For compactness, in future, we will write statements such as this as $P_b(G(\phi, w)) > 1 - \delta$, where P_b means probability, using belief state b as the measure on w.[1]

3.5 Observations

We use as observations the locations and normals of hand contacts, as well as the swept path of the robot as it moves through free space. By comparing the hand

[1]In computing such probabilities we are making the implicit approximation that the probability is constant over each grid-point-centered cell of the grid, which is in sharp contrast to our assumption of smooth underlying probability functions. We are also making the assumption that the boundary of the goal region lies on the boundaries between grid cells.

contact locations and normals and the robot's swept path to the surface and volume occupied by the object at each state in our belief grid, we can calculate how likely it is that the object was actually in that state. For the experiments reported here, we use the expected/observed hand contact locations and normals both while tracking the current belief state and while searching for actions, but use the swept path only for tracking the current belief state.

In this framework, one could also use many other types of sensors as observations, such as fingertip "pre-touch" IR sensors, laser rangefinders, further estimates of object location from vision, or any combination of sensors that allow us to compute the relative probabilities of the states in our belief-state grid.

3.6 Actions: World-relative trajectories

Our goal is to select among possible robot motions online, based on sensory information incorporated into the belief state. It is typical, in lower-dimensional control problems such as mobile-robot navigation, to use a uniform discretization of the primitive action space. Such a fine-grained discretization of the primitive action space for a robot with many joints presents two problems: first, there is a large branching factor in the choice of actions; second, the horizon (number of "steps" that must be made before the goal is reached) is quite long, requiring significant lookahead in planning to select an appropriate action.

Instead, our strategy will be to generate, off-line, a relatively small set of *world-relative trajectories*. A *world-relative trajectory* (WRT) is a function that maps a world configuration $w \in \mathcal{W}$ into a sequence of Cartesian poses for the robot's end effector. In the simple case in which w is the pose of an object, then a world-relative trajectory can just be a sequence of end-effector poses in the object's frame. Given a WRT τ and a world configuration w, the sequence of hand poses $\tau(w)$ can be converted via inverse kinematics (including redundancy resolution) into a sequence of via-points for the arm in joint-angle space. So, if we knew w exactly and had a valid WRT for it, we could move the robot through the hand poses in $\tau(w)$ and reach the desired terminal configuration of the arm with respect to the object. The first point on every trajectory will be the same "home" pose, in a fixed robot-relative frame, and the robot will begin each trajectory execution by moving back to the home pose ϕ_h.

In general, we won't know w exactly, but we will have to choose a single w to use in calculating $\tau(w)$. Let $w^*(b)$ be the world state w for which $b(w)$ is maximized; it is the most likely state. We can execute $\tau(w^*(b))$, and have the highest probability of reaching the desired terminal configuration according to our current belief state. We command the robot to follow the trajectory by executing *guarded move* commands to each waypoint in the sequence, terminating early if a contact is sensed. An early contact (or reaching the end of the trajectory with no contact) results in an observation that can be used to update the belief state. In addition to the collision point, we obtain further contact observations by carefully closing the fingers when a collision is sensed.

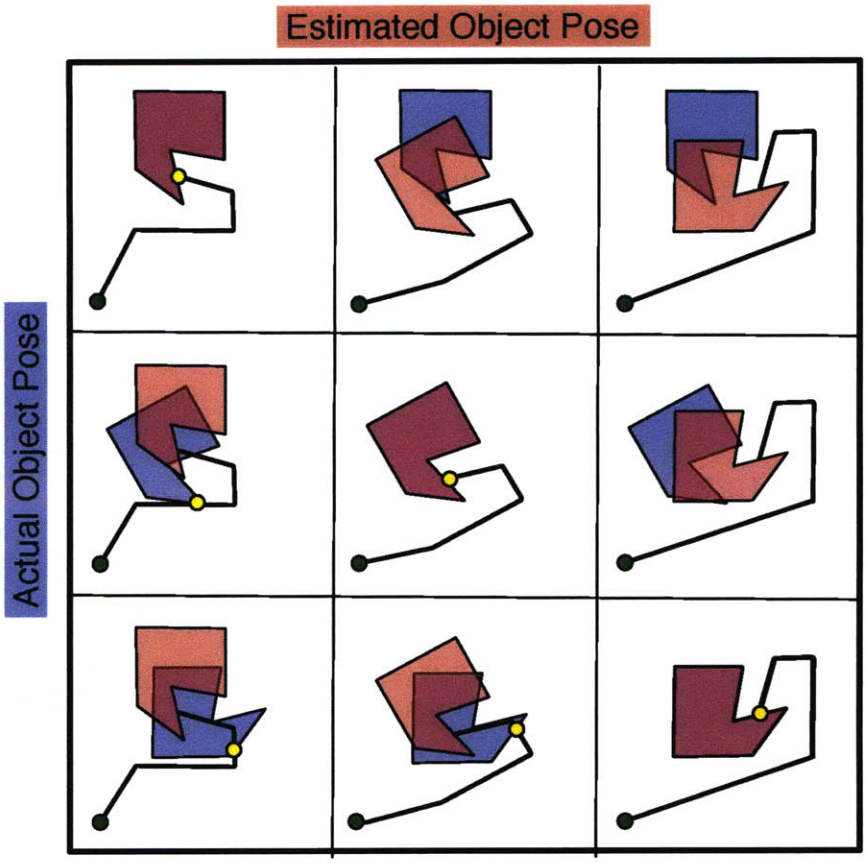

Figure 3-3: The $\Omega_\tau(w, e)$ matrix for a WRT τ.

One way to think of WRTs is as temporally extended "macro actions." This choice of actions allows our forward search to have a relatively small branching factor, and results in effective goal-directed action even with limited lookahead.

3.7 Belief-state update

In order to track the current belief or do forward search in the belief space, we need to be able to update the belief state grid based on the action taken and the observation received. Belief-state updating is a straightforward instance of Bayesian filtering, as explained in section 1.2. The following sections will explain the observation and transition models that are used while doing the belief update.

3.8 The observation and transition models

The observation and transition models allow us to predict the observations we might expect to see given the action taken, by specifying $P(o|s', a)$ and $P(s'|s, a)$. Much of the information about the observations we expect to see (for the range of w we

might encounter) can be pre-computed for each τ and stored, so that it need not be re-calculated every time we wish to consider τ as a possible action to take next. During the on-line execution phase, we will use the stored observation information, together with the continually updated belief state, to select and execute appropriate trajectories.

Each τ is characterized by *feasibility* and *contact observation* functions. Each estimated pose e induces a different actual trajectory $\tau(e)$ in robot coordinate space. The feasibility function for τ, $F_\tau(e)$, is true if trajectory $\tau(e)$ is kinematically feasible for the robot, and false, otherwise. The observation function for τ, $\Omega_\tau(w, e)$, is indexed by an actual world configuration w and an estimated world configuration e, specifying what would happen if the trajectory $\tau(e)$ were executed in world w; that is, if the robot acted as if the world were in configuration e, when in fact it was in configuration w. The observation function $\Omega_\tau(w, e) = [\phi, c]$ specifies what contacts, if any, the robot will sense during execution of $\tau(e)$ in w, where ϕ is the Cartesian position of the robot hand relative to e when the contact occurs (or reaches the end of the trajectory), and c is the set of local sensor readings that can be expected when a sensed contact occurs (and has the form of a list of contacts that contain both the location on the hand and the contact normal, as would be seen by the robot hand sensors, or 'none', if no sensor readings would be seen). The swept path of the robot is all of $\tau(e)$ in the event that the robot reaches the end of the trajectory without contact, and $\tau(e) - \phi$ (the trajectory up to the point of contact) in the event that the robot makes contact.[2]

Figure 3-3 shows the $\Omega_\tau(w, e)$ function for a WRT τ and a space of 3 world configurations, and how it is determined. Each row corresponds to a different true pose (x, y, θ) of the object in the world (w), which is drawn in blue. Each column corresponds to a different estimated pose of the object (e), which is drawn in red. On the diagonals, the true and estimated poses are the same, so the figures lie on top of one another. The estimated pose e determines the trajectory $\tau(e)$ that the robot will follow (in this case, our robot is a point robot in x, y). The trajectories are shown in black. Each one starts from the same home pose, shown in green, and then moves to a sequence of waypoints that are defined relative to the estimated pose of the object. Yellow circles indicate situations in which the robot will make contact with the object. It happens on each of the diagonal elements, because the nominal trajectory makes contact with the object. In the elements in the bottom-left part of the figure, there is a contact between the robot and the actual object during the execution of the trajectory, before it would have been expected if the estimated pose had been the true one. In the elements in the upper right part of the figure, the trajectory terminates with no contact. In all cases, the observation gives information about the object's true location, which is used to update the estimated pose.

The observation model describes the sensory conditions (e.g., finger contacts) that can result from a given action in a given state. In an off-line process, for each WRT τ, we construct a representation of the observation function, $\Omega_\tau(w, e)$, on the

[2] $\tau(e)$ is a trajectory and ϕ is just a single pose, so the two are not of the same type, to be subtracted directly. By $\tau(e) - \phi$ we simply mean the trajectory with the segment from the point of contact on removed.

discretized w, e space. In the case of a single object with a canonical support surface on a table, the space of w and e is characterized by the (x, y, θ) coordinates of the object (although the approach can also be applied more generally).

3.8.1 Computing the observation matrix

Computing an entry of the observation matrix requires simulating a trajectory forward from a starting robot pose, and calculating if and when it contacts objects in the world, and, if it does, what the nominal sensory readings will be in that situation. This simulation includes closing the fingers when a collision is detected, to gather additional contact information. This is a geometric computation that can be done entirely off-line, relieving the on-line system of performing simulations.

This computation may seem prohibitive, since for a x, y, θ grid of just 31x31x25 = 24,025 points, having to simulate all combinations of w and e in pairs would require $24,025^2 = 207,792,225$ simulations. However, the crucial insight here is that if trajectory $\tau(e)$ is kinematically feasible and there are no other objects nearby, then the observation depends only on the relative transformation between w and e, as shown in Figure 3-4. For two sets of w and e with the same relative transformation, as with the examples in the figure, w and $\tau(e)$ may differ, but $\Omega_\tau(w, e)$, which is expressed relative to e, is the same. Thus, when calculating the full $\Omega_\tau(w, e)$ matrix for a WRT τ, we can pick a single e (for instance, the initial $w^*(b)$), compute $\tau(e)$, and simulate just that sequence of robot poses while varying w. The number of simulations required to compute $\Omega_\tau(w, e)$ is therefore merely the number of points in the belief grid that have nontrivial probability, and running them takes just a few seconds. Details of doing these simulations are contained in Appendix C. Once the simulations are completed, the results can be stored for fast re-use when selecting actions on-line.[3]

3.8.2 Computing kinematic feasibility

In practice, there are times in which $\tau(e)$ will not be kinematically feasible, or there can be other obstacles that would collide with $\tau(e)$. Assuming that any obstacles such as the table have known locations as well as known geometry (for uncertain obstacle locations we could use a conservative estimate), $F_\tau(e)$ depends only on e and not on w (since $\tau(e)$ only changes with e, and different w can only stop our trajectory short), and so we could remove the object from our simulation entirely and calculate $F_\tau(e)$ for variable e.

However, just as we have only one actual belief state at any time, we have only one $w^*(b)$, and thus only one e that we need to worry about during execution. Thus, we can defer the calculation of $F_\tau(e)$ for the τ under consideration until we are trying

[3]In our implementation, the uncertain part of the belief state is limited to rigid transformations, for which no two e translate to the same $\tau(e)$. If the belief state included, for instance, reflected object poses, situations could arise in which two e had the same $\tau(e)$. If that were the case, $\tau(e)$ would only needed to be simulated once.

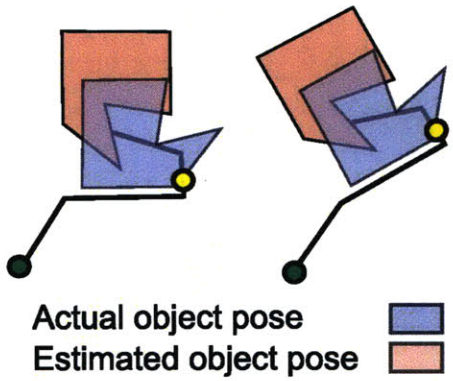

Figure 3-4: Predicted observations depend only on the relative transformation between the actual and estimated object pose.

to select the next action, at which point we can calculate just $F_\tau(w^*(b))$ for each τ and eliminate infeasible actions from consideration.

3.8.3 Observation probabilities

Given a $\tau(e)$ and a resulting observation $[\phi, c]$ (either the predicted nominal observation from the $\Omega_\tau(w, e)$ matrix, or the actual observation made during execution), we need to specify $P(\phi, c|w, \tau(e))$, the probability of observation $[\phi, c]$ given that the object is at w, for each w in our belief state grid. ϕ represents the pose of the robot hand relative to e at the end of executing the trajectory $\tau(e)$, and c represents the contacts and normals seen by the hand at that end pose. These probabilities are necessary for the belief-state update, both while tracking the current belief state and while planning through the POMDP forward search tree. There are two cases to examine:

Case 1: The robot reaches the end of the trajectory with no contact

In this case, ϕ will be at or very near the endpoint of $\tau(e)$, and c will be $None$. In this case, $P(\phi, c|w, \tau(e))$ is the probability that the robot executing $\tau(e)$ will not contact the object placed at w, or in other words, the probability that the path swept out by the robot does not contain the object. The robot can have significant error both in control (trying to follow the trajectory) and in proprioception (reporting what trajectory was actually taken), and so ideally we would consider, under our model of both types of error, all the possible trajectories that the robot might actually take when trying to execute $\tau(e)$. In the case of an actual observation made during execution, we can record the trajectory actually taken as reported by the robot's proprioception, and thus not have to worry about the control error, but proprioception error remains a problem. For each trajectory the robot might have taken, we would like to compute whether it would have contacted the object placed at w. We could then integrate the probabilities of the trajectories that do not contact to find $P(\phi, c|w, \tau(e))$. If T is the

set of all possible trajectories τ the robot might have taken under our control and proprioception error models, and NC is an indicator variable that is 1 if τ contacts w and 0 otherwise:

$$P(\phi, c|w, \tau(e)) = \int_{\tau \in T} NC(\tau, w) P(\tau; \tau(e)) = P_{free}(w, \tau) \ .$$

Integrating over all possible trajectories would be excessively difficult, particularly since the trajectories could, in theory, deform arbitrarily. However, under the simplifying assumption that all τ in T are of the same shape as $\tau(e)$, but just shifted and/or rotated, we could sample possible likely trajectories and record whether they contact or not, as in part a) of Figure 3-5. This assumption is typically reasonable for the scenarios we encounter, because although the proprioception error of our arm can be significantly different for different trajectories (especially, for instance, ones that approach from above versus ones that approach from the side), within the short segment of a single trajectory that could plausibly be in collision with an object at any pose, the effect of the proprioception error can typically be approximated reasonably by a constant shift. We can model the probability of a particular trajectory, $P(\tau; \tau(e))$, shifted a distance $s(\tau)$ from the nominal trajectory $\tau(e)$ as:

$$P(\tau; \tau(e)) = \mathcal{N}(s(\tau); 0, \theta_p^2) \ ,$$

where $\mathcal{N}(s(\tau); 0, \theta_p^2)$ is a zero-mean Gaussian with variance θ_p^2 depending on our robot's proprioception error, evaluated at $s(\tau)$.

When using trajectory samples, if our set of sampled trajectories is T_s, we could estimate $P_{free}(w, \tau)$ as follows:

$$P_{free}(w, \tau) = \frac{\sum_{\tau \in T_s} NC(\tau, w) P(\tau; \tau(e))}{\sum_{\tau \in T_s} P(\tau; \tau(e))} \ .$$

Because even sampling in this fashion with enough samples to generate reasonable collision probabilities would still be quite expensive, we make the drastic simplifying approximation of trying to compute the max instead of the integral. If T_{nc} is the set of trajectories τ for which $NC(\tau, w)$ is 1:

$$P_{free}(w, \tau) \propto \max_{\tau \in T_{nc}} P(\tau; \tau(e)) \ .$$

These probabilities are unnormalized, but we will normalize the belief-state grid after doing the observation update.

For any state w for which the nominal trajectory $\tau(e)$ does not make contact, the nominal trajectory is the most likely trajectory, and so $\max_{\tau \in T_{nc}} P(\tau; \tau(e)) = \mathcal{N}(0; 0, \theta_p^2)$. If we estimate the deepest point of collision, d, between the nominal trajectory's swept path and the object placed at w, as shown in part b) of Figure 3-5, we know that $\tau(e)$ would have to be shifted by at least distance d in order to be non-colliding. Thus:

$$\max_{\tau \in T_n c} P(\tau; \tau(e)) \leq \mathcal{N}(d; 0, \theta_p^2) \ .$$

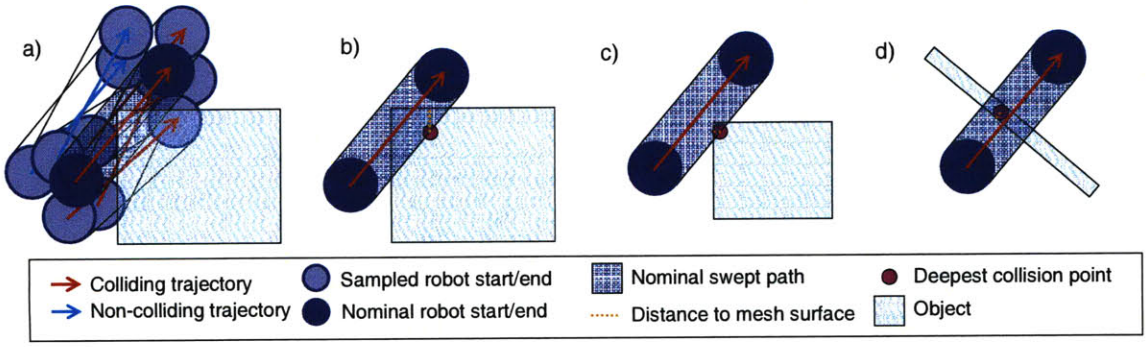

Figure 3-5: Swept path diagrams: a) Sampling possible trajectories for collisions. b) The deepest collision point in the nominal swept path. c) and d) Two very different situations that are indistinguishable when using the deepest swept path collision point

Therefore, we use $\mathcal{N}(d; 0, \theta_p^2)$ as an optimistic estimate of $P_{free}(w, \tau)$. This approach has its drawbacks; in particular, the two situations in parts c) and d) of Figure 3-5 have similar d, even though the situation in part c) should have a fairly high probability of non-collision while the situation in part d) should have an approximately zero probability of non-collision. This drawback makes it difficult to generate accurate observation probabilities for very thin objects. However, in general, using $\mathcal{N}(d; 0, \theta_p^2)$ as an estimate of $P_{free}(w, \tau)$ is a reasonably fast and useful way to use the swept path of the robot to eliminate w that would have collided significantly with $\tau(e)$.

Case 2: The robot contacts the object

The robot stops at the point of first contact along the trajectory, and so the probability of seeing $[\phi, c]$ along $\tau(e)$ is the probability of not seeing contact until the point ϕ along the trajectory, seeing contact at ϕ, and then given that there is contact at ϕ, seeing the particular contact observation c:

$$P(\phi, c|w, \tau(e)) = P_{free}(w, \tau(e) - \phi) P_{contact}(\phi|w) P(c|\phi, w, \tau(e)) \ .$$

The probability of not seeing contact until the point ϕ is the same as the probability that the trajectory up to that point is free, or $P_{free}(w, \tau(e) - \phi)$, and is calculated in the same way as P_{free} from case 0. $P_{contact}(\phi|w)$ depends on the proprioception and contact sensor location error of the robot, and the distance between the robot and the surface of the object when placed at w. Even if the pose ϕ would not nominally contact the surface of the object at w, it is possible that the robot is not exactly where it thinks it is, and so it might have made contact at that pose anyway. If $\hat{\phi}$ is one possible actual pose of the robot when it observes that it is at ϕ, we could

compute the last two parts of $P(\phi, c|w, \tau(e))$ as follows:

$$P_{contact}(\phi|w)P(c|\phi, w, \tau(e)) = \int_{\hat{\phi}} P(c|\hat{\phi}, w, \tau(e))P(\hat{\phi}|\phi)SC(\hat{\phi}, w) \ ,$$

where $SC(\hat{\phi}, w)$ is an indicator variable that is 1 when $\hat{\phi}$ exactly contacts the surface of the object at w and 0 otherwise. However, this integral would be excessively difficult to compute exactly, and so we will make the drastic simplifying approximation, again, of trying to compute the maximum instead of the integral. If $\hat{\phi}_s$ is the set of poses for which $SC(\hat{\phi}, w)$ is 1:

$$P_{contact}(\phi|w)P(c|\phi, w, \tau(e)) \propto \max_{\hat{\phi}_s} P(c|\hat{\phi}_s, w, \tau(e))P(\hat{\phi}_s|\phi) \ .$$

Because this would still require extremely expensive calculations involving the geometry of both the hand and object, we make the additional simplifying assumption that contact between the hand and object can only be made at the points on the hand where the sensors actually saw contact. This allows us to consider just the geometry of the hand and a disembodied contact location, c_l, and normal, c_n, in the global frame. If the point on the surface of the object that is most likely to have caused the observed contact position and normal has a global position of p_l and a surface normal of p_n, we can use our robot's proprioception and contact sensor location error models to estimate the probability of the observed contact position being at c_l instead of p_l, and we can use our model of the contact sensor normal error to estimate the probability of the observed contact normal being c_n instead of p_n.

The observed contact point could have been caused by contact with any of the faces of the mesh, and so ideally we would compute the maximum as follows:

$$\max_{\hat{\phi}_s} P(c|\hat{\phi}, w, \tau(e))P(\hat{\phi}|\phi) = \max_{f \in F} P(c_n|f_n)P(c_l|f_l)$$

$$where \ \ P(c_l|f_l) = \mathcal{N}(||c_l - p_l||; 0, \theta_l^2)$$
$$and \ \ P(c_n|f_n) = \mathcal{N}(||c_n - p_n||; 0, \theta_n^2)$$

and where f is a face in the set of mesh faces F, with f_l being the closest point on f to c_l and f_n being the outward normal of f, θ_n is the standard deviation of the sensor normal, and θ_l is the standard deviation of the sensor contact location (including proprioception noise).

However, our object meshes contain too large a number of faces to compute probabilities for all of them; searching over a small set of nearby faces would be tractable, but we do not do so in this implementation. Instead, as a fast approximation we assume that the contact was caused by contact with the closest mesh face to c_l, f^*. This maximizes $P(c_l|f_l)$ but not necessarily $P(c_n|f_n)$. Parts a) and b) of Figure 3-6 show f_l and f_n for contact locations inside and outside an object. In these two cases, the closest mesh face is the one that maximizes the product $P(c_n|f_n)P(c_l|f_l)$. This is not always the case, however. Part c) shows a case where our fast approximation is

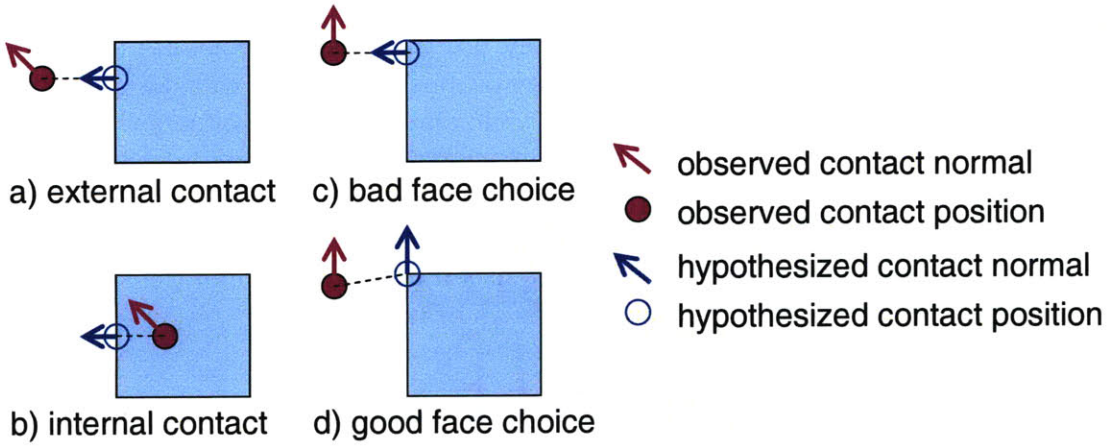

Figure 3-6: Nearest-contact diagrams: a) Nearest surface contact to an observed contact outside an object. b) Nearest surface contact to an observed contact inside an object. c) A case where picking the nearest face to the contact causes a poor approximation of the most likely contact normal. d) The most likely contact position and normal that should have been chosen in c.

a bad one, since the closest mesh face (the left side of the box) has a very different normal from the contact, whereas the top of the box, which is only slightly farther away, has an identical normal. The f_l and f_n that should have been used are shown in Part d).

Computing probabilities over the belief-state grid

We sample the continuous function $P(\phi, c|w, \tau(e))$ using our approximations at the grid locations, to find the observation probabilities for the entire belief-state grid, then normalize over the grid. If there are multiple observed contact points in our observation c, we assume that they are independent, and thus we can compute their probabilities separately and then multiply the probabilities before normalizing the grid. For observations in which there are multiple finger contacts obtained by closing the fingers at the pose of initial contact, we can treat each finger movement as an independent action that takes place after the arm motion is finished. One 'action' is then treated as four independent actions for the purposes of calculating observation probabilities: one in which the arm moves along the trajectory until the first contact is made, one for the first finger closing until it makes contact (or until it closes all the way), with the hand fixed at the first-contact pose, one for the second finger closing, and one for the third finger closing. The path swept out by the arm and by each finger-action, and the corresponding observed contact(s), are used to compute $P(\phi, c|w, \tau(e))$. We also treat these probabilities as being independent, and thus they can also be multiplied together before normalizing the grid.

Currently, we only calculate $P_{free}(w, \tau)$ (the swept path probability) for the actual observations received during execution, for use in tracking the current belief

state. While searching through the POMDP forward search tree, we let $P_{free}(w, \tau)$ be 1 for all states in the belief grid, and only use the expected contacts to compute $P(\phi, c|w, \tau(e))$. This makes clustering observations (which we will describe in the next section) easier, since many different w yield similar sets of contacts that provide similar amounts of information about the object pose, whereas the information contained in the swept path varies a great deal depending on the absolute pose of the object. This makes our current estimates of the effects of actions rather incomplete, but because contacts generally provide more useful information than the swept path, we are still able to choose reasonable actions anyway.

3.8.4 Reducing the observation branching factor

When describing the POMDP search tree, we said that we would branch on the possible observations that would occur given a start belief state and an action. However, even though we have reduced the number of simulated outcomes from all pairs of (w, e) to only those w in the start belief state with nontrivial probability, for a grid size of 24,025, we could have up to 24,025 different observations. This is because the observation includes both the Cartesian pose of the hand when it stops relative to e, as well as the contacts made with the object at w, and so exact duplicates only arise when the hand stops in the same place upon first contact with the object at w, and closing the fingers results in seeing the exact same contact positions and normals. (Even so, there are often duplicates, just not enough of them to reduce the number of observations significantly.) This is entirely too large a number to branch on, so we must reduce the number of observation branches significantly. To do so, we cluster our observations and represent each cluster with a single canonical observation, and then only branch on the canonical observations.

Because our actions are largely only for information-gathering, and because we are using receding horizon planning, it is not crucial that we be able to choose the optimal action at each time step, since most actions will bring us closer to our belief-state goal, and we will be able to re-plan in the next time step anyway. Choosing actions that are closer to optimal merely brings us to our belief-state goal faster. Thus, we do not need to take into account all possible outcomes, merely enough of them to give us a reasonable picture of the likely effects of each action. This allows us to cluster and prune our observations in a fairly aggressive fashion, resulting in an extremely small number of observation branches. Our clustering and pruning method has four phases, which we will explain in turn.

Initial clustering by similar contacts

We can cluster observations with similar contacts, since we are only concerned with the information to be gained about the object pose, not about its actual location. Since we are not using the swept path in our search tree predictions, observations that have similar contacts give us similar amounts of information. For instance, in Figure 3-7, even though the contacts made along the trajectory are in different absolute positions, and are at different points along the side of the object, both outcomes tell us

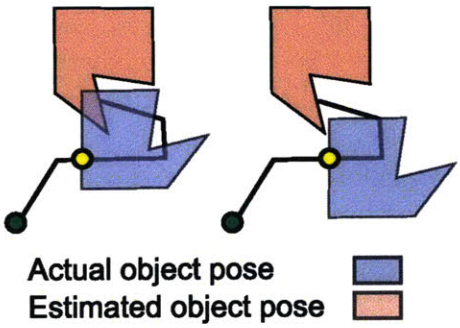

Figure 3-7: Observations with similar contacts can be clustered.

the same thing: that one of the sides of the object is at the observed contact location, with a normal in approximately the observed contact normal direction. These two observations would result in similar resulting belief states, and thus would have similar branch values, and so we cluster them together. Each cluster of observations is represented by a canonical observation.

When clustering observations, we start the first observation in its own cluster, and add each successive observation to that cluster if it is similar enough to the canonical observation (updating the canonical observation each time to be the most likely observation in the cluster), starting new clusters when an observation is too dissimilar. Two observations are deemed sufficiently similar if there is a one-to-one matching between the two sets of observed contacts between the hand and the object. For each pair of contacts, a match is found if: the same hand sensor is observing the contact, the contact positions on the object are within a fixed distance from each other, and the sensed normals are within a fixed angle from each other.

Initial pruning of unlikely clusters

In the next step, we discard canonical observations that are highly unlikely to ever arise during the entire course of grasp execution. For instance, an observation cluster that contains only the observations from two grid points at the very edge of our initial belief state would only be observed if the true world state were at one of those two far-flung poses relative to our estimated world state. Such an observation might have only a probability of .00004 of being observed in our first action. After the first action, our uncertainty will only get narrower as we learn more about where the object actually is, making the probability of seeing that observation drop further. Thus, we can discard such unlikely observations. We cannot, however, just discard observations with low probabilities in the initial belief state, because once we have narrowed down our belief state to a small region around the actual object position, the nominal observations may have low probability in the initial belief state but near-unity probability when the uncertainty is very small. For instance, an observation cluster that contains only two grid points that happen to be at the center of our grid (where w is very close to e) would also have a probability of .00004 in our initial belief

state, but if we have narrowed our belief state to knowing exactly where w is, the probability of seeing one of those two observations will be close to 1.

Thus, we recalculate the probabilities of each observation at multiple levels of uncertainty (in our implementation, we use three levels: the start Gaussian uncertainty, one where the standard deviations for each dimension are 1/5 the original levels, and one where the uncertainty has been narrowed to within .5 cm and .05 radians), and discard any observations that are highly unlikely to occur (probability of less than .01) at all three levels of uncertainty. Alternatively, for a smoother estimate of the likelihood of encountering an observation contained in any particular cluster, we could inversely weight each grid point's probability by the distance of that grid point's w from the center point (which is e), then add up the weighted probabilities belonging to each cluster. This would make points close to e appear to have high probability while points far from e would appear to have lower probability.

This pruning step brings our canonical observation set down to 68, which is a more reasonable number of observations to store and re-load along with the contact observation probabilities, each time we wish to grasp the object using that WRT. As mentioned earlier, we use the contact observations but not the robot's swept path to compute the observation probabilities for constructing the POMDP search tree. Thus, the observation probabilities that we store with each cluster do not take into account information that could be gained through the robot's swept path.

Further clustering

Pruning still leaves us with too many observation clusters, so we can further cluster canonical observations directly by similarity of resulting belief states. If two observations have similar effects on the belief state, we would like to cluster them into a single canonical observation. Thus, we cluster resulting belief states that are similar in both entropy and covariance, simply by adding each belief state in turn to existing clusters until they don't fit in any of the existing clusters, then starting new clusters, just as in the first clustering step. In particular, if belief state i has an entropy of e_i and variances in x, y, and θ of v_{xi}, v_{yi}, and $v_{\theta i}$, respectively, we cluster belief states i and j if the entropies are within a small threshold ($|e_i - e_j| < \alpha_e$, where α_e is a small percentage of the maximum grid entropy, which in our implementation is 10%), and if for each axis, the variances for that axis either have a ratio close to 1, or are within a small threshold of each other. For instance, for x: $1/\alpha_x < v_{xi}/v_{xj} < \alpha_x$, or $|v_{xi} - v_{xj}| < \beta_x$, where α_x is a constant greater than and near 1 (1.2 in our implementation), and β_x is a small constant (1 cm in our implementation). Belief states with similar entropies and covariances are similar in information content, and thus we expect that they have similar values and can be put in the same observation branch.

Examples of such clusters are shown in Figure 3-8, where the belief states are generated by doing a belief update with the canonical observations from each outcome on the initial belief state. (Although our belief state is three-dimensional, these grids are summed over theta for easier visualization.) As you can see, some of these belief states are actually fairly dissimilar, and yet, we can see that the clustered observations yield qualitatively similar amounts of information. Because this clustering process

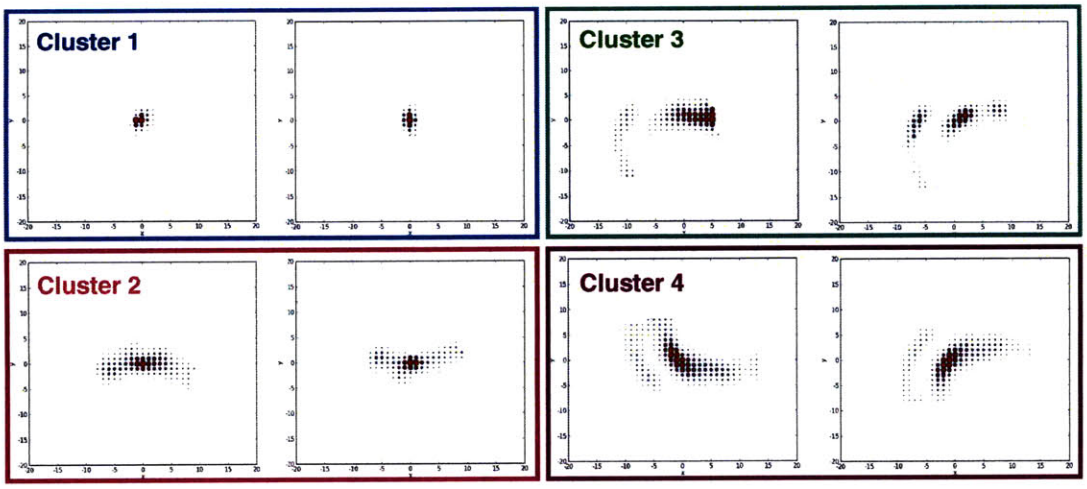

Figure 3-8: Observations with similar resulting belief states can also be clustered. This figure shows four pairs of belief states that are clustered together.

does not take very long, it could easily be done on-line for the current belief state, although we currently only do this step off-line, based on the initial belief state. For the initial belief and our powerdrill goal-seeking grasp example, this brings our canonical observation set down to 26. If we wished to incorporate the swept path information in our observation probabilities without fracturing the clusters, it would be easy to re-compute the observation probabilities with the swept path included after this final clustering step, but we do not currently do so.

On-line pruning

Even 26 is still too large a branching factor, so as a coarse approximation, when setting up the POMDP search tree on-line, we sort observation branches for each action by their probability and only add them, most likely branches first, until the cumulative probability rises above a fixed threshold. The probability of each observation branch is found by adding up the probabilities in the current belief state associated with the grid points whose observations belong to that cluster (an example is given in section 3.11.2). In our experiments, that threshold is set to .5; it was experimentally determined that considering probabilities above that threshold did not improve performance significantly. Of course, the threshold can be raised or lowered depending on computation time constraints. This reduces our observation branching factor in our powerdrill goal-seeking grasp to only 2 branches; typically there are between 1 and 7 branches remaining.

When calculating canonical observations on-line, it is crucial to re-calculate the probabilities of each observation given the current belief state, as well as to re-select the most likely observation as the representative observation, since both change significantly as the belief state changes. An example showing observation clusters, their stored observation grids, and how to use them is given in Appendix 3.11.2.

3.8.5 The transition model

The transition model captures the dynamics of how the object moves in response to contact with the robot during execution of a trajectory, $\tau(e)$. If no contact was made with the object, it is assumed that the object does not move. If contact was made with the object, we would ideally like to predict where the object in each starting pose w could have ended up at the next time step (w') by integrating over all possible trajectories the robot could have actually taken, and for each trajectory, looking at what contacts could have been generated by that trajectory, and where those contacts might have moved the object given that it could have rotated about any point in its support polygon, and given a range of possible friction coefficients. In practice, trying to make predictions of that sort would be intractable. Instead, we try to make a simple prediction of where the object might have been bumped, and then assume that the object is either somewhere near where it started, or else somewhere near our predicted bump location.

The transition model is thus a mixture of Gaussians around two outcomes: 1) the object remains approximately in place, and 2) the object gets bumped by the contacts that were made when the trajectory terminated. To compute the most likely translation and rotation for the bump outcome for each w, we use ϕ and c (observed or predicted) to compute the object-relative contact positions and normals on an object placed at w, ignoring the fact that the contact locations may not be on the surface of the object. (States for which the contact locations are nowhere near the object surface will be ruled out in the observation update.) A unit vector is placed at each contact position, in the direction of the observed normal. The object is assumed to rotate about its center of mass, and the forces and torques are summed. The object is then assumed to translate a fixed distance per unit force in the direction of the sum total force, and it is also assumed to rotate a fixed rotation per unit torque. We denote the bump outcome pose estimated in this way as $w_b(w, \phi, c)$.

Since these motion estimates are unreliable, we assume that an object starting in exactly pose w will transition to poses near w (for the outcome where the object remains in place) and to poses near $w_b(w, \phi, c)$ (for the bump outcome) with probabilities proportional to Gaussians centered about w and $w_b(w, \phi, c)$. The standard deviations of these Gaussians are calculated based on a parameter that dictates the probability that the object stays within 1 cm and .1 radians of the nominal position. As the size of the expected bump in the bump outcome grows, we lower the probability of leaving the object within the 1 cm/.1 radian zone, since if we expect the object to move a great deal, but are uncertain about the direction and magnitude of movement, we expect the possible range of motion to spread out farther in all directions.

More concretely,

$$P(w'|w, \tau(e)) = (1 - p_b)\mathcal{N}(\delta(w, w'); 0, \sigma_s^2) + p_b\mathcal{N}(\delta(w_b(w, \phi, c), w'); 0, \sigma_b^2) \ .$$

where p_b is the probability of being in the bump outcome, σ_s is the standard deviation of the Gaussian centered about w, σ_b is the standard deviation of the Gaussian

centered about $w_b(w, \phi, c)$, and $\delta(w, w')$ is the normalized Euclidean distance between w and w' (with 1 cm being equivalent to .1 radians in our implementation).

For each w in the grid, we sample $P(w'|w, \tau(e))$ for the w' at each grid point location up to a fixed distance away (2 cm and .2 radians, in our implementation) from either w or $w_b(w, \phi, c)$. Because we are only sampling probabilities at grid locations, not taking integrals over cells, these probabilities will not sum to 1, but since we are only concerned with the relative probabilities of each grid point, we can simply normalize the grid when we are done adding up all the contributions for each w' from each starting w on the grid.

In using the observed or predicted observation to compute the likely bump transition, we are over-weighting the observation, since the same observation is used to compute the transition probabilities as to compute the observation probabilities, which are then multipled together. However, only a small portion of the information contained in the observation is used to compute the transition probabilities, and the Gaussians about both outcomes are chosen to be large enough to ensure that we are unlikely to be much more confident than we ought to be given the observation. An alternate way to think about the transition and observation updates is that the states are spread out in the transition update enough to cover the bump outcome in all directions, and then the observation consists of two parts that are modeled as being independent. The first observation component, in the form of the estimated total force and torque generated by the contacts seen, is first used to perform the transition update on the belief state, and then the rest of the observation, in the form of the swept path of the robot and the locations and normals of a set of contacts that should coincide with the object's surface, is used in the normal observation update as described above.

3.9 RRG-Goal: Robust execution of goal-seeking WRTs

In many situations, the strategy of executing a single goal-seeking WRT over and over again until our belief state goal condition is met is sufficient to robustly grasp an object. Consider the execution of a single WRT with the goal of grasping the object, and let the system be in initial belief state b. We first compute the most likely state, $w^*(b)$, and generate $\tau(w^*(b))$, a sequence of waypoints in robot coordinates. We then command the robot to move to each waypoint in turn using guarded moves, collect the resulting observation (either early contact or successful termination of the WRT), and use that observation to update the belief state. If the goal condition on the belief state is not satisfied, we retrace the previous trajectory back to the home pose (to avoid any further contacts, which we will not be able to predict effectively), relativize τ to the new most likely world state, and re-execute. Repeated execution of the goal-seeking WRT relative to the current most likely state is continued until the belief state goal condition is met, at which point the robot declares success and attempts to pick up the object. We refer to this simple algorithm for robust execution of a single,

goal-seeking WRT (with the addition of allowing an optional reorientation action to be used if the goal-seeking WRT is infeasible, which will be described shortly) as the *RRG-Goal* algorithm, short for Relatively Robust Grasping with Goal-seeking WRT.

Figure 3-9 shows the operation of the system while grasping a rectangular box using the RRG-Goal algorithm. The robot attempts to execute a grasping trajectory, relative to the most likely element of that belief state. The first image in the top row shows where the robot thinks the most likely state of the box is relative to its hand. The second image shows the location of the robot at the first waypoint in that trajectory: we can see that the object is actually not in its most likely position (if it were, the robot's hand would be centered above it). The third image in the first row shows the initial belief state, with probabilities depicted via the radius of the balls shown at grid points on the (x, y, θ) state space of the box. Subsequent belief state images show the belief state summed over θ for easier visualization, as in the fourth image in the first row. In action 1 (row 2), the hand executes a guarded move toward the next waypoint in the trajectory, and is terminated by a fingertip contact on the corner of the box, as shown in the first figure in the second row. The middle figure in the second row shows the probability of observing that fingertip contact in each world configuration. Combining this information with the initial belief state, we obtain the updated belief state shown in the third figure in the second row. It is clear that the information obtained by the finger contact has considerably refined our estimate of the object's position.

The third and fourth rows of figures show a similar process. The same WRT is executed a second time, now with respect to the most likely state in the updated belief state. This time, the hand is able to move all the way down, and the fingers close on the box, with the resulting belief state shown in the final figure. Now, given a goal condition such as having the box centered between the fingers within 1.5 cm and oriented within 10 degrees of being straight, but not being concerned where along the box the fingers are grasping (shown by the oval in the updated belief state), we can evaluate the probability that it holds in the current belief state. In this case, it holds with probability $\geq .8$, so if δ were .2, we would terminate.

With only one WRT, we do not need tree search to select an action. Even with the addition of an optional reorientation action, we would only use such an action if the goal-seeking WRT were infeasible, so there is still no need for tree search.

3.10 Additional WRT actions

The most basic strategy of executing a single goal-seeking WRT can be effective in many situations, as demonstrated in the experimental results. However, there are many situations in which it may fail. The two main problems are kinematic infeasibility and lack of information. To deal with these problems, we will add additional WRT actions and use lookahead search to select among them.

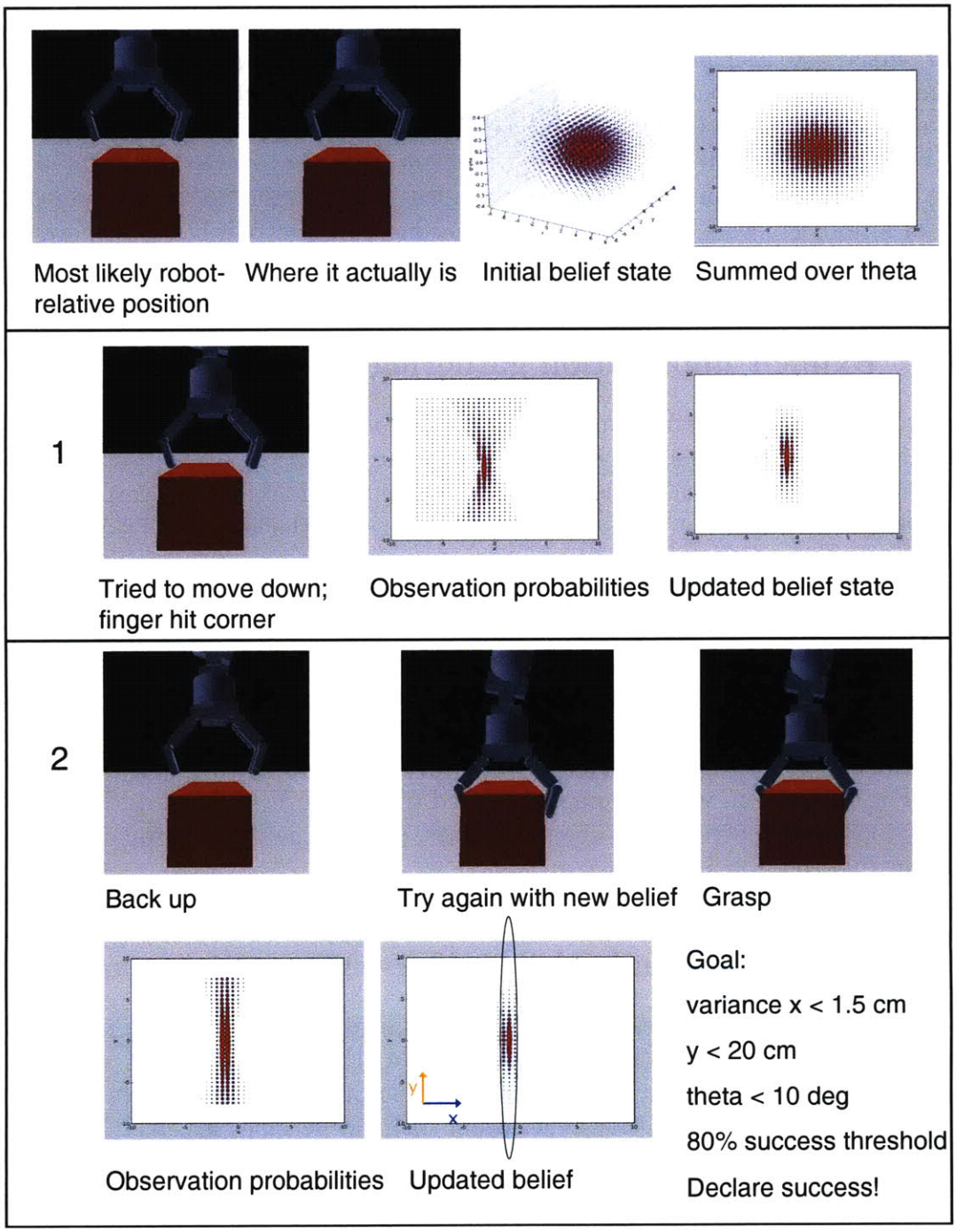

Figure 3-9: Execution of WRT and (x, y, θ) belief state update.

3.10.1 Goal-seeking trajectories

In our experiments, we use only a single, hand-generated goal-seeking WRT for each object. However, for some tasks, we may need more than one goal-seeking WRT to effectively achieve the goal for all possible object locations; there is no reason we need to limit ourselves to a single goal-seeking WRT. Kinematic constraints arising from the boundaries of the workspace may render execution of $\tau(w)$ simply infeasible for some values of w. It may also be that, even when it is executed in the appropriate world state ($\tau(w)$ is executed in w) a collision will result, due to obstacles. In addition, if our goal condition contains noncontiguous regions corresponding to different possible grasps of an object, we would need more than one WRT to be able to achieve the different goal grasps.

3.10.2 Explicit information gathering

The execution process described in the previous section will succeed if the goal-seeking trajectories result in local sensory observations that provide enough information to update the belief state so that it is eventually concentrated around the correct world state. However, this will not necessarily happen. Consider the final belief state shown in Figure 3-9. It is clear that the object is well localized in the x and θ dimensions, but there is still considerable uncertainty in the y dimension (the grasp that the robot executed knows only that the fingers are on the box in the y dimension, but not where along the box they are). If the goal had required that the box be grasped very near the center in the y dimension, for example, then this result would not have satisfied the goal condition, we would have no real way to improve the situation through further execution of our goal-seeking WRT, and the control loop would run forever. For this reason, we will sometimes need to execute WRTs that are designed explicitly to reduce uncertainty in the belief state, not to achieve the ultimate desired world configuration.

3.10.3 Reorienting and other actions that move the object

We can also execute trajectories that attempt to change the actual state of the world, rather than merely trying to reduce uncertainty in the belief state. If all of our goal-seeking trajectories are kinematically infeasible in $w^*(b)$, for instance, we may wish to reorient the object so that at least one of them becomes feasible. To do so, we can add a WRT that attempts to grasp the object (using a grasp that does not necessarily satisfy the goal conditions) and then rotates the object after successfully running to completion. Of course, such actions can also cause unpredictable object movements if $w^*(b)$ is not the same as w, or sometimes even if it is the same. However, as long as we can compute the transition and observation models for an action and perform appropriate belief updates, that action fits into our framework. This could also include actions that shove the object, perhaps against a known surface that constrains the belief space in one or more dimensions, such as in [1].

In our experiments, we have reorientation grasps for objects for which the goal-seeking trajectory can be out of reach; these reorientation grasps are all grasps from above that attempt to rotate the object about its center of mass. The observation model is the same for reorientation grasps as for other WRTs; however, the transition model additionally models the movement of the object, taking into account the possibility of the rotation failing or leaving the object at an intermediate rotation. Also, because reorientation is more likely to fail if we do not know where the object is, the probability of the rotation failing is estimated to be higher when the covariance of the current belief state is larger.

3.10.4 Generating WRTs

The previous section describes the various types of WRTs that we would like to have. How can we generate a good set of WRTs? We divide the problem into generating WRTs for goal achievement and for information gain.

Generating reasonable goal-oriented trajectories is a task-dependent problem. It could be done by explicit teaching or using a general grasp planner [3, 38] to generate goal-seeking grasps, and calling a robot motion planner to generate trajectories that achieve the grasps. Then, re-expressing the trajectories in world-relative coordinates will allow them to be used relative to other world configurations in which they are kinematically feasible. In the experiments reported in this thesis, we use a single goal-oriented trajectory constructed through demonstration (a person moves either the simulated or the real robot through the intended trajectory, and recording waypoints manually).

Most of the information-gain trajectories used in this thesis were also constructed through demonstration, although a few were constructed automatically. Information-gain trajectories can be constructed automatically by considering different major surfaces of the objects in the world and planning trajectories to touch those surfaces; or, to find face pairs that are graspable and to construct trajectories to grasp them. In our implementation, we constructed information-gain trajectories by finding hand positions that place the fingers on nearly parallel pairs of object surfaces. We then found collision-free trajectories to those hand positions, for the nominal goal-seeking object position, using the default planners in the OpenRAVE motion planning system [14]. (Each trajectory took at most a few tens of seconds to plan.) We also added an information-gathering WRT for most of the objects that just sweeps horizontally across the entire workspace looking for the object, which is particularly useful for roughly locating the object under high uncertainty.

Intuitively, for each object that we wish to grasp, we need to provide a set of information-gathering and goal-seeking WRTs that constrain the possible object states in such a way that the constrained object position is not easily confusable with any object position outside of the goal condition, within the start uncertainty. The contacts for the single box-grasping WRT in our example in section 3.9, for example, only constrain the box in one direction; when all the nominal contacts are made, the box is still free to slide along its length. Thus, if our goal region requires us to know where the fingers are along the length of the box, even if we execute the one grasp

and make the resulting contacts a large number of times, there would still be many object poses outside the goal region that could be the true pose of the object. For an entire set of WRTs, we can ignore the geometry of the hand and pretend that to execute a WRT is to observe a disembodied set of nominal contacts. If we place the object in its nominal pose, and pretend that we are observing the nominal contacts for all of the WRTs in our available set at once, under the assumption that there is no noise in the system (which is equivalent to observing those nominal contacts an infinite number of times in the presence of Gaussian noise), those contacts should constrain the object pose such that no other object pose outside the goal region and within our initial uncertainty can still be likely. If it is the case that some object pose outside the goal region is likely, that means that if the object were exactly where we thought it was initially, and we executed our entire set of WRTs in turn an infinite number of times, our resulting belief state would still not allow us to say that the object were within the goal region with sufficiently high probability.

Furthermore, if we expect the workspace to be cluttered, and have geometric models of the obstacles (such as could be provided by a laser rangefinder), we can provide additional WRTs that would allow us to collect sufficient information even when some of our WRTs are infeasible due to possible collisions.

3.11 RRG-Info: using POMDP forward search to select WRTs

Given a set of available WRTs, including information-gathering, goal-seeking, and reorientation WRTs, we would like to select WRTs in a sequence that allows us to reach our belief state goal condition in as few actions as possible. Intuitively, if we are highly confident that we know exactly where the object is, we would like to execute a goal-seeking WRT, which will allow us to declare success if, at the end of the trajectory, our belief state is consistent with the WRT having succeeded with high probability. If we don't know where the object is with sufficient confidence, we would like to gather information about its position, and if all of our trajectories are kinematically infeasible or in collision given where we think the object is, we would like to attempt to reorient it. All of these conditions can arise automatically from a planner that is merely trying to maximize total reward, by assigning values to specific belief states, and by using POMDP forward search, as described in section 3.2.

We refer to the use of POMDP forward search to select among a full complement of information-gathering, goal-seeking, and reorientation WRTs as the *RRG-Info* algorithm, short for Relatively Robust Grasping with Information-Gathering.

3.11.1 Static evaluation function

We have not yet specified the form of the static evaluation function that will assign values to belief states at the leaves of our POMDP search tree, $v(b)$. These values are propagated up the search tree, as described in section 3.2, and used to estimated the expected value of taking each action under consideration. We can use any function we

want, but there are two obvious functions we might consider: entropy and probability of success.

Entropy

In characterizing how good our belief state is after carrying out an action τ, we might decide to use the entropy of the belief state as a measure of how much information we have gained by executing τ. We can let $v(b) = H_{max} - H(b)$, where H_{max} is the maximum entropy we can have on our belief state (found when probabilities are uniform) and $H(b)$ is the entropy of the current belief state. Intuitively, higher values of $v(b)$ correspond to lower entropy in our belief state, and thus more information about the location of the object; we can expect that lower entropy belief states are more likely to result in success, and so picking the action that most lowers the entropy brings us closer to succeeding.

Probability of success

While entropy is a reasonable measure of how well we have narrowed down the position of the object, and is in fact shown to work fairly well in [22], we can calculate a much more direct measure: the probability of succeeding, if we were to execute one of our goal-seeking WRTs in that state, and terminate. More specificially,

$$v_b(b) = \max_{\tau_g}[P_b(G(\Omega(\tau_g(w^*(b)), w)))]$$

which is, according to our belief state b, the maximum probability that G will be true in the state resulting from executing any of our goal-seeking actions using the most likely state, $\tau_g(w^*(b))$.

This measure has a distinct advantage over the entropy measure particularly when the goal is asymmetric in the dimensions of the belief state, since using entropy will cause one to choose actions that reduce uncertainty unnecessarily in dimensions we may not care about. For instance, in our box example in section 3.9, if we only care that the fingers are around the box, and not where they are along the length of the box, using an entropy measure takes more actions to achieve the same success level as using probability of success (this is borne out in simulated experiments).

3.11.2 Using the stored observation outcomes

Pre-calculating the observation outcomes off-line is crucial when searching through the POMDP forward search tree, since re-simulating the observation matrix for every branch of the tree while choosing each action on-line would be completely intractable. However, the pre-calculated, stored observation outcomes are computed for only a single e (that being the initial most likely state). During execution, the current most likely state changes as we make observations and perform belief updates. This means that in order to use the stored observation outcomes to predict what will happen under an action $\tau(e)$, we need to either shift and rotate the stored observation outcomes

to be relative to the current e, or else shift and rotate the current belief state to be relative to the initial e used to compute the stored observation outcomes. Shifting either grid results in a loss of precision, as the shifted grid probabilities must be found using interpolation, and if we have not computed extra probabilities outside our normal belief grid size, it can also result in the zeroing out of some grid points near the edge of the grid, for which there are no nearby points in the shifted grid. This is typically not an issue while searching through the POMDP forward search tree, since high levels of accuracy are not required when we are merely trying to estimate the value of choosing among a small set of actions to gather information. On the other hand, for tracking the current belief state based on the actual observations seen by the robot during execution, we may wish a greater level of accuracy. Fortunately, running the simulations to calculate a single observation matrix for the τ that was actually run (and only for those w with non-trivial probability) does not take too long, and so we can improve our accuracy while tracking the current belief state by calculating the observation probabilities for the actual observation on-line.

Figure 3-10 shows the setup for an example of using clustered observation outcomes and their stored observation grids. In this example, the robot is grasping a box, and has access only to a single goal-achieving WRT. The initial belief state has negligible uncertainty in θ, low uncertainty in y (where the hand is along the long side of the box), and very high uncertainty in x (where the box is from left to right in the picture). In action 1, the robot thinks the box is directly under the hand, so it moves down and grasps, but the box is actually far to the right, and so the hand misses entirely and sees no contacts. Because the swept path of the robot rules out all the x positions in the center of the grid, the updated belief state looks like the grid in the top right of the figure, with the most likely state being to the left of the ruled-out region. (The left mode is slightly more likely than the right due to the asymmetry of the hand; one side has two fingers while the other side has only one.) The rest of the figure shows the observation outcomes for the goal-achieving WRT for action 2. To execute the next action, the hand backs up and moves to the start of the WRT relative to the new most likely state.

In this figure and the ones to follow, pictures of the robot arm grasping the box have several visualizations of the corresponding belief state. The dark blue rectangle shows where the robot thinks the most likely state of the box is currently. The light blue rectangles show the object poses at one standard deviation around the mean of the belief state; these given an indication of the variance of the current belief state. Graphs of the belief state, grid points, or observation likelihoods are red in high-probability areas, and fade to black in lower-probability areas; white areas have zero or near-zero probability.

There are three observation clusters remaining that have non-negligible probability, shown in Figure 3-11. In outcome 1, the box is far to the right (this is the outcome that actually happens), and so the hand expects to miss entirely again. In outcome 2, the box is centered sufficiently in the hand to allow it to grasp successfully and collect three fingertip contact observations, and in outcome 3, the box is too far to the left, and so the left finger will hit the top of the box, resulting in a single fingertip contact observation.

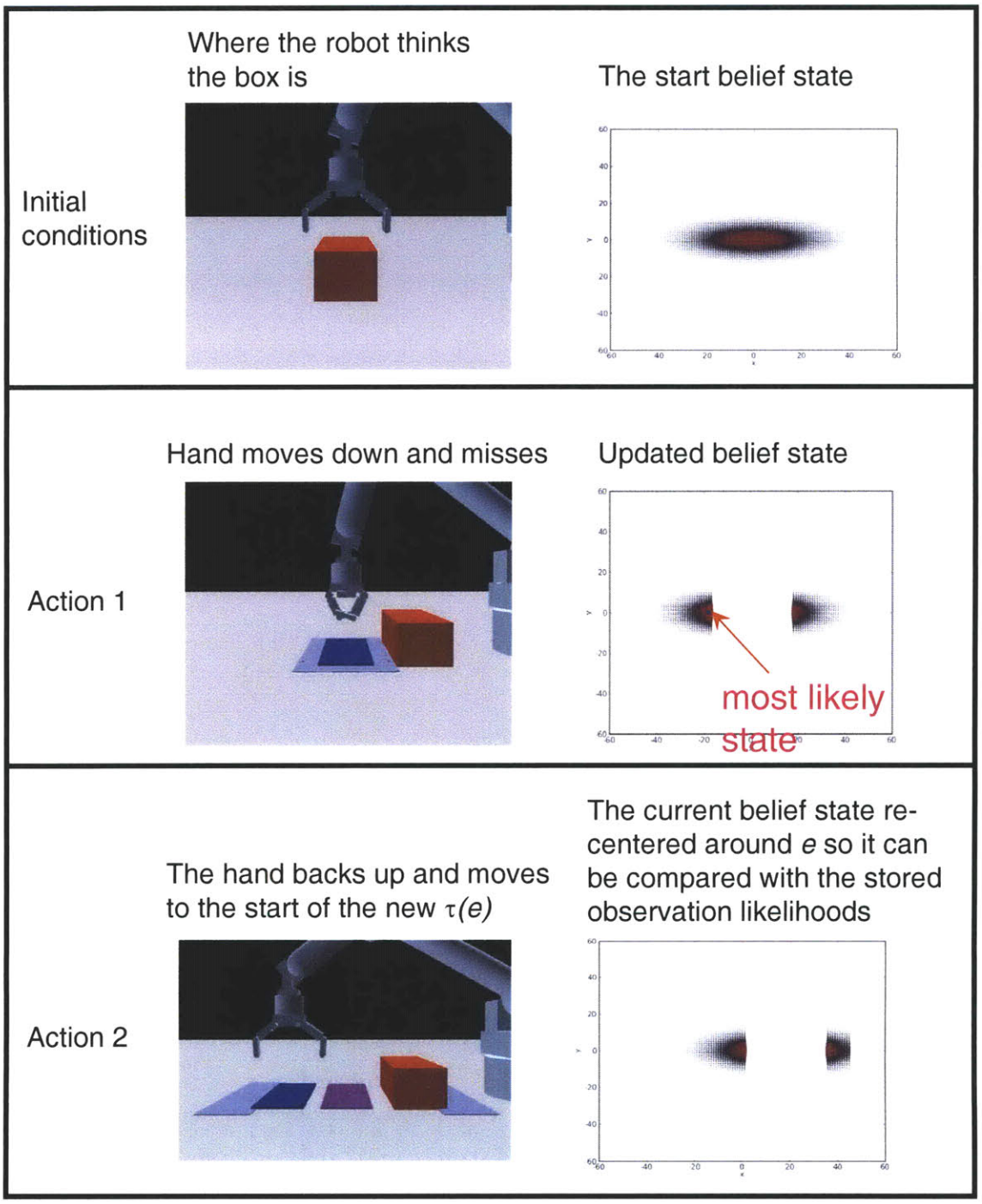

Figure 3-10: Setup for a box grasping example demonstrating clustered observation outcomes.

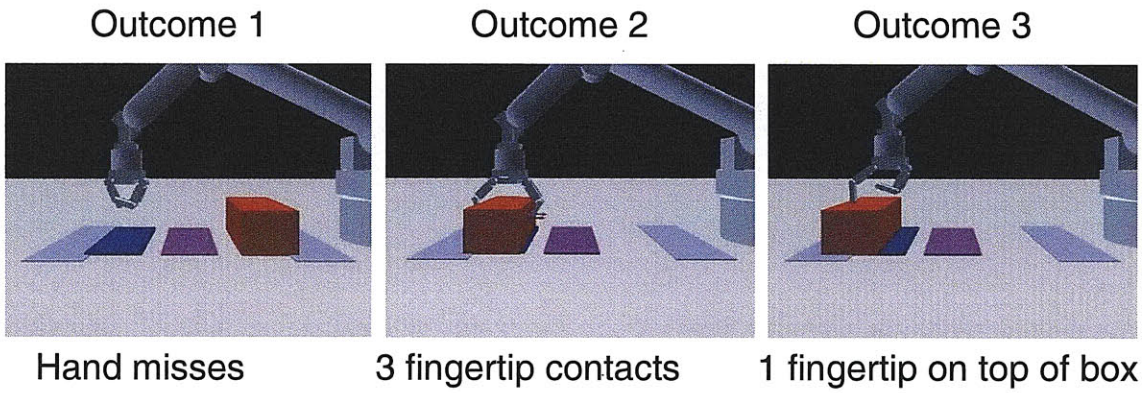

Figure 3-11: The canonical observations for each of the three clustered observation outcomes.

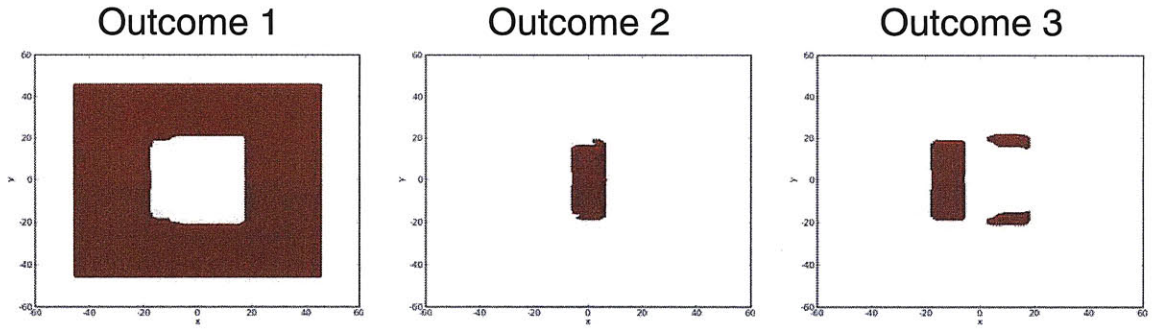

Figure 3-12: The grid points that belong to each cluster (computed off-line and saved).

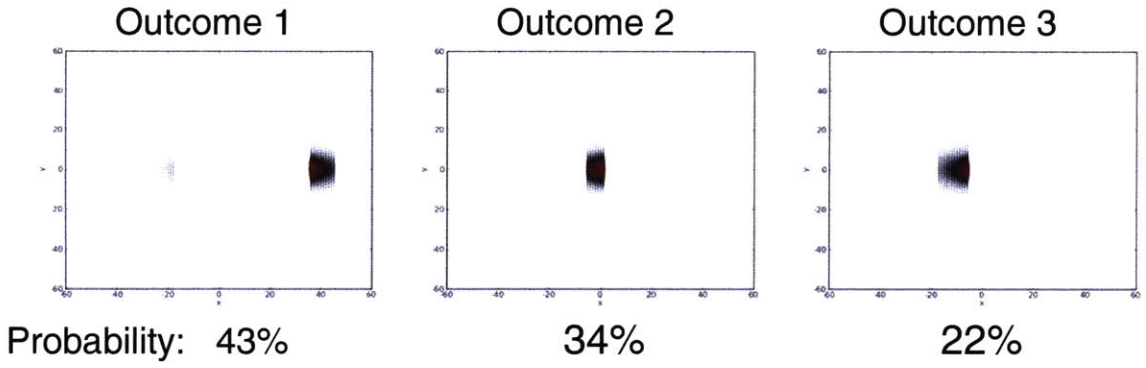

Figure 3-13: The parts of the current belief contained in the grid points that belong to each cluster. The probabilities are summed to obtain the outcome probability.

Figure 3-12 shows the subsets of the grid that belong to each observation cluster, computed off-line with respect to the initial most likely state (the center of the grid). The first grid contains all the box positions in which the hand misses entirely. The second grid contains all the box positions in which the hand either surrounds the box or at least touches one of the sides of the box (since touching at least one side is sufficient to localize the box in x). The third grid contains all the box positions in which only one fingertip touches the top of the box. Since the hand has three fingers, that means either the thumb (the lone finger) touches the top, or one of the other fingers touches the top while the other finger misses (in order for one finger to miss while the other touches, the box must be shifted a fair amount in the y-direction). One other outcome that one might expect to see, in which two fingers touch the top of the box, has been ruled out by the swept path in action 1.

In order to compute the current probabilities of the grid points that belong to each cluster, a copy of the current belief state is shifted so that the current grid likelihoods are expressed relative to the initial most likely state, as shown in the rightmost grid in the third row of Figure 3-10. The parts of the belief state that are too far to the right to fit in the shifted grid are essentially zeroed out, and thus are not considered in the observation branches. While this can cause problems when the belief is highly bimodal and with modes spread far apart, in practice, any action that attempts to grasp the object in the current most likely mode is likely to rule out that mode through the swept path when executed. Since the current tracked belief state still contains the other mode, it will then become the new most likely mode in the next time step. Since we are using a receding horizon planner and can select a new action based on this new information, the negative consequences of cutting off far-away parts of the belief state are minimized.

Figure 3-13 shows the portions of the shifted belief state that belong to each cluster. The probabilities in each cluster are summed to obtain the current probability of seeing an observation in that cluster, which are the probabilities used for the observation branches in the POMDP search tree.

Figure 3-14 shows the stored observation grids for each outcome (also relative

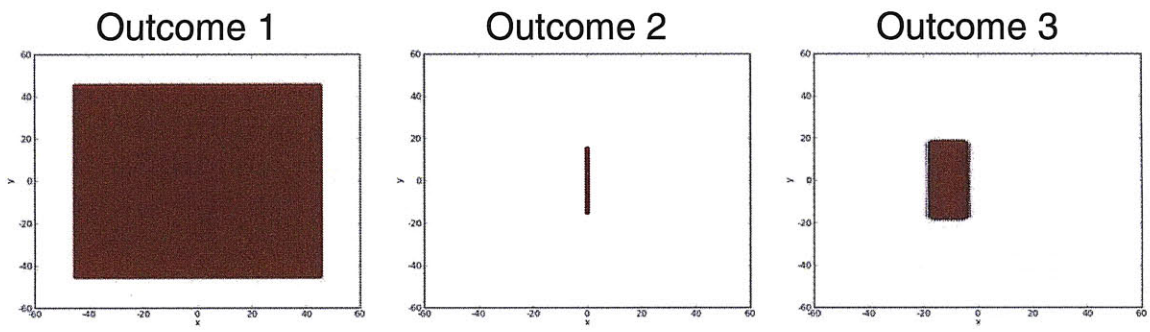

Figure 3-14: Probabilities for the canonical observation for each outcome (computed off-line and saved).

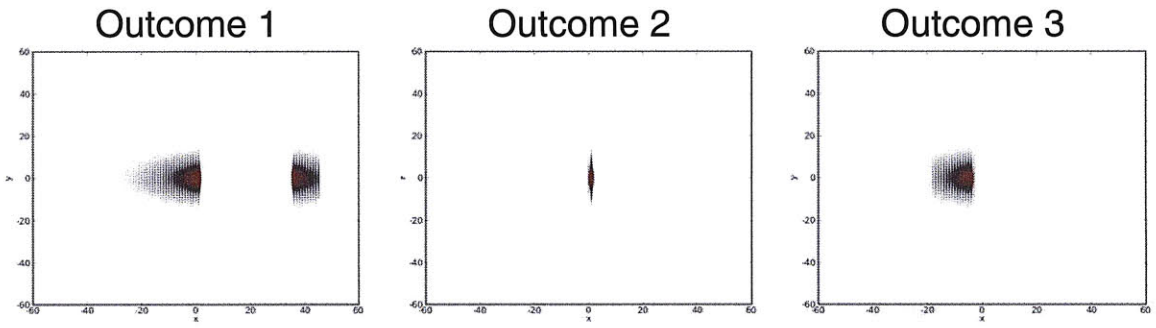

Figure 3-15: The predicted resulting belief state for each outcome.

to the initial most likely state at the center of the grid). These grids contain the observation probabilities for the nominal observation for each outcome, and are used to do belief update when creating the POMDP forward search tree. Note that the nominal observation probabilities are very different than the grid points contained in each outcome, which were shown in Figure 3-12. The easiest way to see the difference is to imagine two separate clusters that have nearly identical observations (which hopefully in our implementation would have been clustered together, but imagine that they were not). The two clusters would have nearly identical stored observation grids, but disjoint sets of grid points in each cluster. The stored observation grids for the identical clusters would assign high probabilities to both sets of grid points from both clusters, since one is indistinguishable from the other given the observation, and might also assign high probabilities to grid points not in either cluster, that also happen to have a high probability of producing the nominal observation.

In Figure 3-14, outcome 1 has uniform probabilities over the grid, since the stored observation grids do not incorporate swept path information. Outcome 2 shows that having three fingertip contacts narrows the likely states to a thin band in x. Outcome 3 shows that having a single fingertip contact on top of the box restricts the box location to an area equal to the top surface of the box.

Figure 3-15 shows the result of multiplying the stored observation grids by the

current (shifted) belief state, which results in the updated belief state, under the assumption that the object does not move when touched. (While doing a real belief update, while constructing the POMDP search tree or tracking the current belief state, we assume that the object does move with some probability, and thus also do a transition update, not shown here.)

3.11.3 Depth of search and goal horizon

Despite cutting down the observation branching factor so drastically, because doing a belief-state update on entire belief grids is a somewhat expensive operation, it is difficult to search farther than a horizon of 3 on the POMDP search tree, and even using a horizon of 3 can take an uncomfortably long time. Fortunately, the goal horizon can be as low as 1–if the starting belief state is sufficiently narrow, or if the robot gets lucky, selecting the goal-seeking grasp and executing it successfully can be done on the very first action. This is very unlike many robot problems that are solved by selecting actions through forward search. For instance, a mobile robot trying to navigate to a goal typically has to perform a number of actions in sequence in order to actually physically reach its goal. Also, with the exception of reorienting the object when it is out of reach, our actions do not intentionally change the underlying state of the world; they only gather information that reduces uncertainty in the belief state. Thus, even selecting actions randomly is likely to eventually get us to the goal belief state, if we do not limit the number of actions the robot can take; searching for an optimal action merely allows us to take fewer actions to reach our belief state goal. Taking fewer actions is, however, highly desirable, since each action takes quite awhile to execute on the robot, and each action we need to take increases the probability of an undesirable outcome such as knocking over the object. If we need 10 actions in order to pick up any object, our algorithm would only be useful for the very patient. If searching more intelligently allows us to reduce the number of actions from 10 to 4, we have gained a great deal of practicality.

3.12 Chapter Summary

In this chapter, we presented a framework for using POMDP forward search to solve 3D grasping problems of objects modeled as triangle meshes. We presented the concept of the *world-relative trajectory* (WRT), which is a robot trajectory expressed relative to the most likely state of the object, and explained the RRG-Goal algorithm, which robustly executes a single goal-seeking WRT while tracking the belief state. We further showed the importance of goal-seeking, information-gathering, and reorientation actions, and also showed how to pre-compute most of the observation model for these WRT actions for fast re-use while selecting the next WRT to execute. We presented the RRG-Info algorithm, which uses POMDP forward search to select among WRTs appropriately in order to reach a belief state goal condition using as few actions as possible. In the next chapter, we will present experiments that test the RRG and RRG-Info algorithms, both in simulation and on a real robot.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

WRT POMDP Experiments

4.1 Experimental Setup

4.1.1 The Robot

We conducted experiments using the WRT POMDP framework with a 7-DOF Barrett Arm and Hand, both in simulation (using Open Dynamics Engine to simulate the physics of the world) and on an actual robot (a 7-DOF Barrett Arm with attached 4-DOF Barrett Hand), shown in Figure 4-1.

4.1.2 Modeling of robot control

Details of how the robot is controlled to execute WRT actions through Cartesian motions are included in Appendix B. Because the robot controllers often do not move the robot exactly along the desired Cartesian paths when executing WRTs, there can be significant control error, both in simulation and on the actual robot. Ideally we would record the path reported by the robot as we execute WRTs and use the path executed instead of the intended path when doing swept path calculations; however, we do not currently do so. Also, because the encoders on the Barrett Arm are at the motors and not at the joints, there is also significant proprioceptive error; the position of the hand can be off by as much as 2 cm at times. We thus model proprioception error when calculating swept path probabilities as having a standard deviation of .5 cm.

4.1.3 Contact Sensors

The hand is outfitted with Nano17 6-axis force/torque sensors at the fingertips, and simple contact pads covering the inside (and some outside) surfaces of the fingers and the palm, as shown in Figure 4-2. The fingertip sensors allow us to estimate the position and orientation of contacts, as detailed in Appendix A. We model the fingertip position error as having a standard deviation of 0.5 cm, and the fingertip normals as having a standard deviation of 30 degrees. Each contact pad is a thin rectangle, approximately 2-3 cm on a side, consisting of two metal plates separated

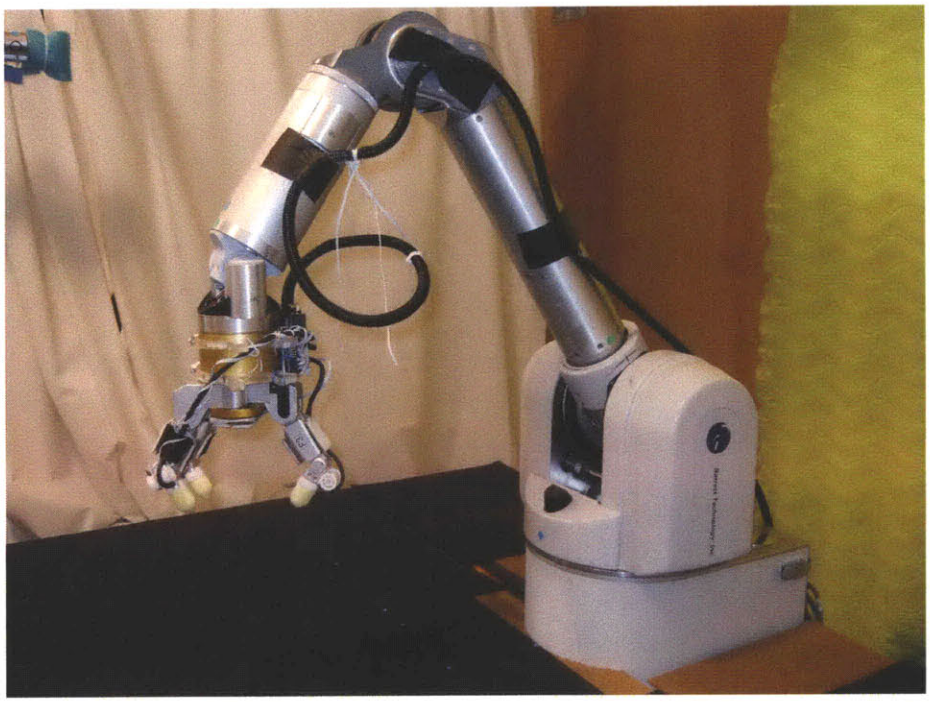

Figure 4-1: 7-DOF Barrett Arm with 4-DOF Barrett Hand.

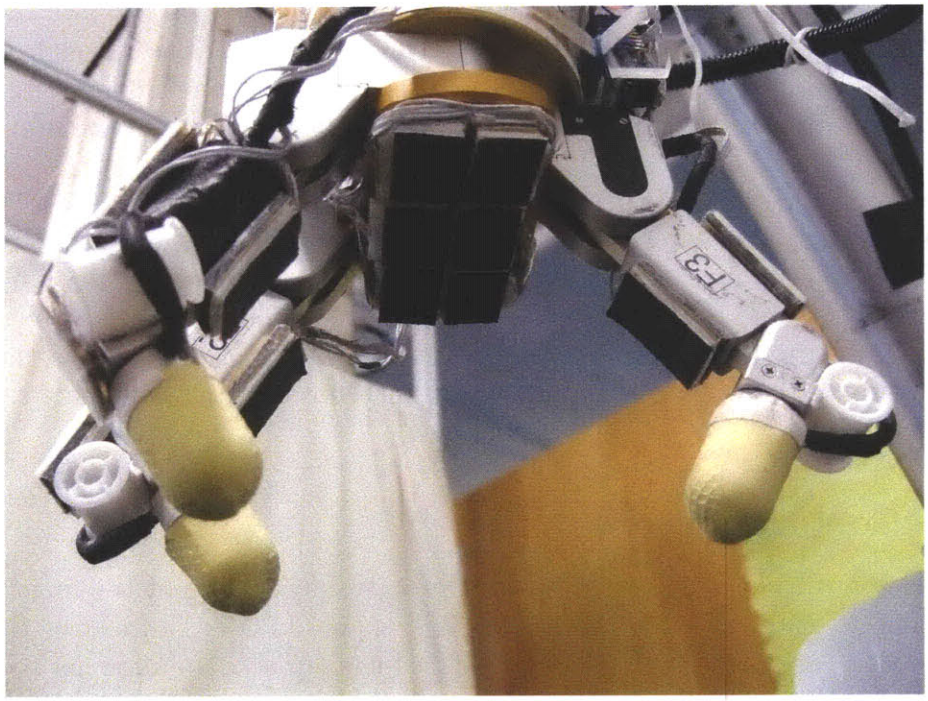

Figure 4-2: Barrett Hand with Sensors.

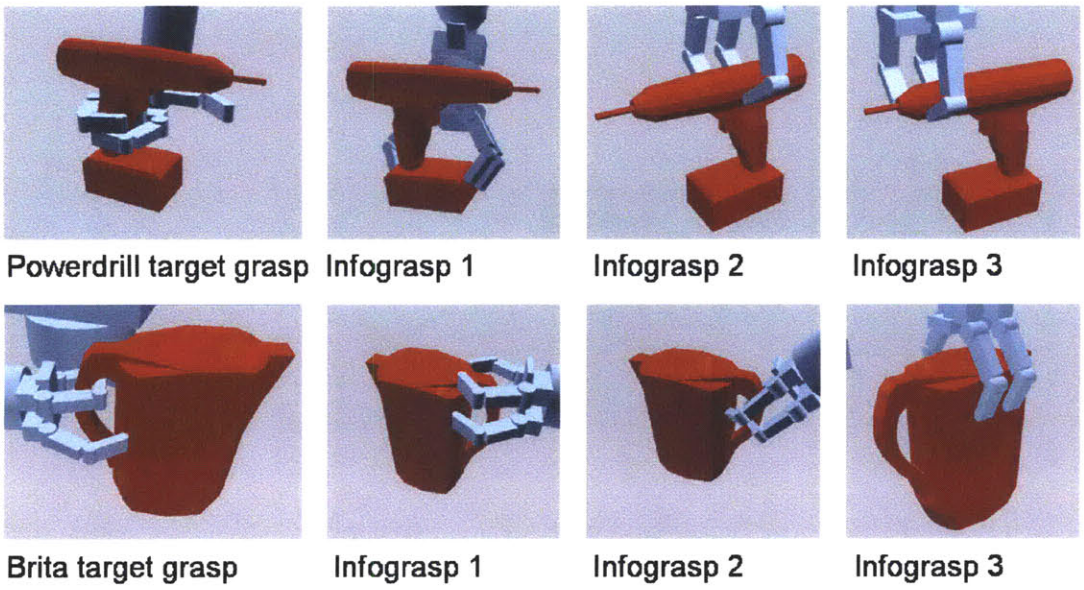

Figure 4-3: WRTs for the Brita pitcher and powerdrill. Infograsp 2 for the powerdrill and infograsp 3 for the Brita pitcher are available in both purely information-gathering and reorientation versions.

by small patches of very light foam, that sense when the metal plates touch. While they are very sensitive to light contacts, they do not provide an estimate of contact location on the pad. Thus, we currently model the location of pad contacts as being the center of each contact pad, with a location error standard deviation of 1 cm, and we model the normal as being the current pad surface normal, with a normal error standard deviation of 90 degrees.

4.1.4 The objects and their goal-seeking grasps

Our experiments used 10 different objects, shown with their goal-seeking grasps in Figures 4-3 and 4-4. Goal regions and required contact locations for each object were hand-chosen to guarantee that being within the goal region ensures a stable grasp of the object. These regions are much larger for some objects than for others. For instance, the goal region for the can is large, since the hand only has to envelop it, and the goal regions for certain objects (the cup, wooden bowl, can, and taperoll) ignore the orientation of the object, looking only at the object position relative to the hand. Other objects (Brita, giant teacup, cooler) have fairly lax position and orientation requirements, but require specific finger contacts to be present in approximately the right places, to ensure that appropriate handle and supporting contacts are made. Still other objects have fairly strict position and orientation requirements, as required by the grasp. For instance, the goal region for the powerdrill is 1 cm in x, 1 cm in y, and 5.8 degrees in θ; any larger, and the hand risks not being able to pull the trigger properly. The full table of goal conditions is shown in Figure 4-5.

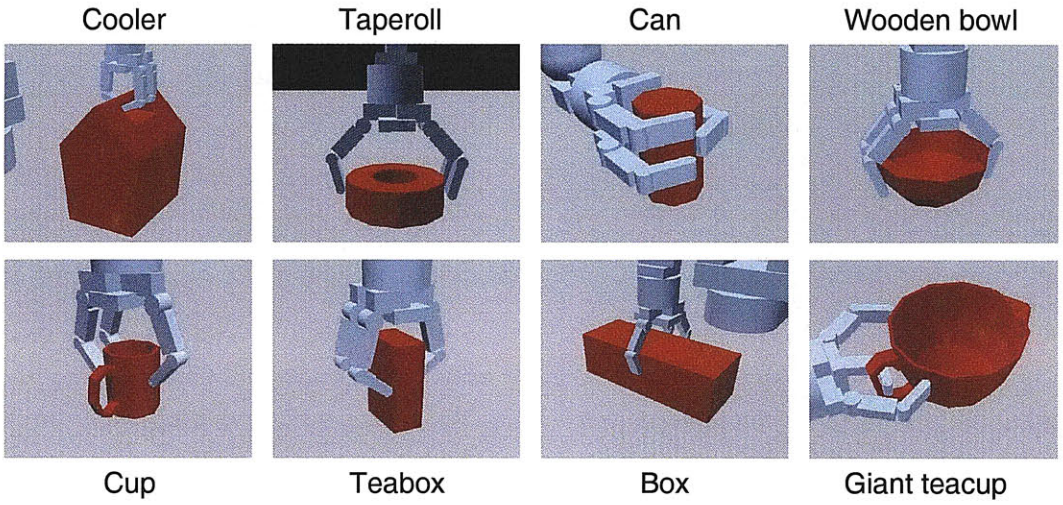

Figure 4-4: The simulated robot grasping the other eight objects.

	cooler	taperoll	can	wooden bowl	cup	teabox	box	giant teacup	Brita	powerdrill
x (cm)	3	2	3	2	1	1.5	1.5	5	5	1
y (cm)	3	2	3	2	1	2.5	1.5	5	5	1
theta (rad)	.15	n/a	n/a	n/a	n/a	.15	.1	.5	.5	.1
contacts	y	n	n	n	n	y	y	y	y	y

Figure 4-5: Goal conditions for all objects.

The planner is asked to try to succeed 90% of time ($\delta = .1$) for all of our experiments. When the belief state meets this criterion, the search and information-gathering process terminates and the robot tries to lift the object. In all experiments, the maximum number of actions allowed was 10; after the 9th action, if the planner had not yet declared success and terminated, the goal-seeking grasp was executed open-loop before evaluating whether the grasp had succeeded. One could alternatively try to maximize the expected utility, by ceasing to take actions when the expected benefit of taking the best action is lower than the cost of taking an action. Although we do use costs and rewards to choose the action with the highest expected reward, we do not use them when deciding when to stop and declare success.[1]

In simulation, it is easy to directly evaluate whether the goal conditions are true, since all positions, orientations, and contacts for both the robot and object are known. For the real robot experiments, goal conditions were evaluated by visual examination of the final grasp. All simulation experiments were carried out with at least 100 trials each.

In our experiments, we compare three different strategies: 1) Open-loop: executing the goal-seeking WRT once, open-loop; 2) RRG-Goal (Relatively Robust Grasping with Goal-seeking WRT only): repeatedly executing just the goal-seeking WRT (and reorienting when the goal-seeking WRT becomes infeasible); and 3) RRG-Info (RRGs with information-gathering): using the full POMDP forward search framework with a horizon of 2 (except where otherwise specified).

It should be noted that 10 actions can take an excessively long time: a 10-action grasp can take over half an hour to run in our current implementation, both in simulation and on the real robot. Although our code is severely non-optimized Python, and thus the robot runs much slower than it might otherwise be able to, having to grasp an object 10 times before picking it up would test one's patience even if it were done more quickly. Because of the large amounts of sensor noise in our system, it is sometimes necessary to grasp nearly that many times to ensure that the object location is sufficiently narrowed down. However, in general, we would like our planners to choose actions wisely, so as few actions as possible are taken.

4.2 Simulation results

Figure 4-6 shows the results for experiments carried out in simulation with initial (Gaussian) uncertainty standard deviations of 1 cm in x, 1 cm in y, and 3 degrees in

[1]If the user would prefer that taking actions be balanced against a lower probability of success in this way, that approach can make sense, and is actually the more natural approach when using POMDPs. However, in trying this approach, two problems were observed. First, when only searching to a horizon of 1 or 2, if it takes 3 or more actions to raise the probability of success to a reasonable level, it is often the case that no actions look worth taking if the costs are not engineered carefully. If we could search to unlimited depths this problem could be reduced, but doing so would take an inordinate amount of time. Second, it seemed much more natural and less frustrating to think of balancing success rates and numbers of actions by stating that we would like this level of robustness, but we are not willing to allow the robot to take more than this number of actions, rather than relying on nebulous costs and rewards for actions, successes, and failures.

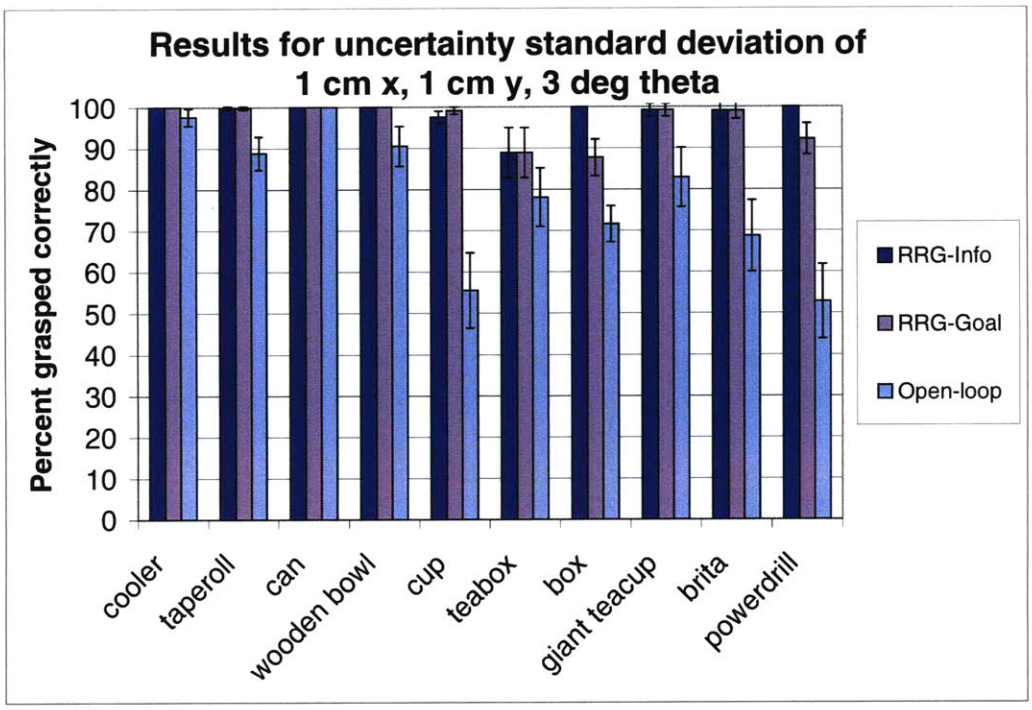

Figure 4-6: Results for all objects at low uncertainty (1 cm x, 1 cm y, 3 deg θ).

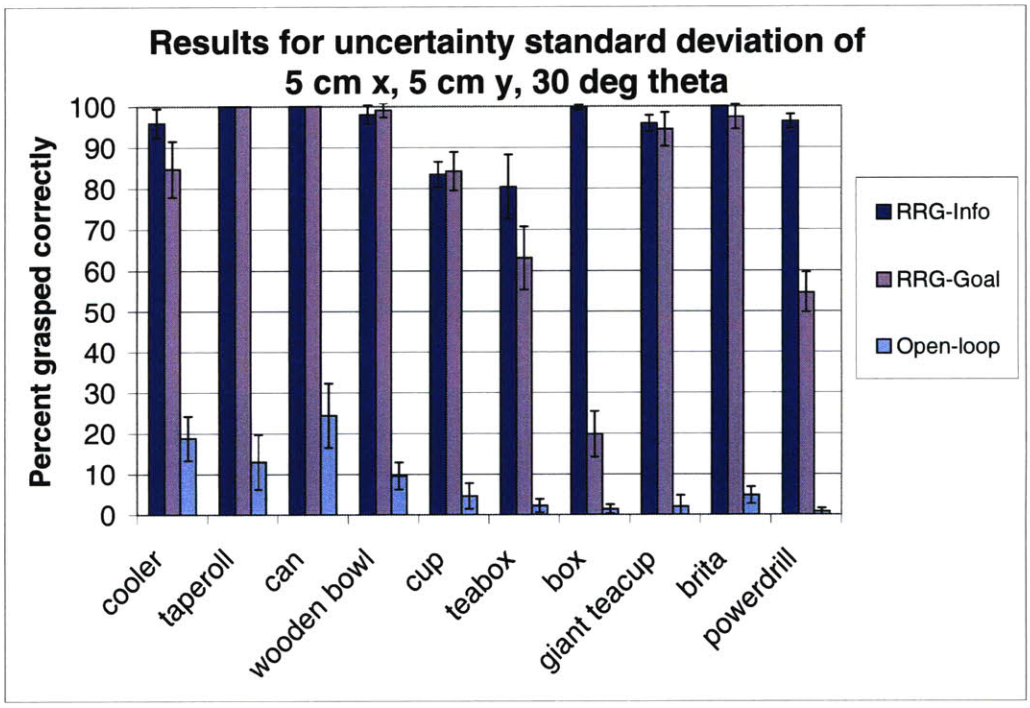

Figure 4-7: Results for all objects at high uncertainty (5 cm x, 5 cm y, 30 deg θ).

θ. The graph shows the percentage of grasps that were executed successfully for each object placed at random positions drawn from the same distribution as the initial uncertainty, for the three algorithms. Even at these low uncertainty levels, executing the goal-seeking grasp open-loop fails with fairly high probability for many of the objects. At low uncertainty levels, RRG-Info only selects the goal-seeking WRT for all objects except the cup, box, and powerdrill, and thus the results are the same for RRG-Info and RRG-Goal for all objects but those three. Using just RRG-Goal allows us to succeed nearly all of the time, and using RRG-Info brings our success rate above 97% for all objects except the teabox. The planner only selects the goal-seeking WRT for the teabox because it recognizes that it will succeed with a probability of approximately 90% (it succeeds 89% of the time) just by selecting the one action. However, raising the target success rate to 95% causes it to select other actions, raising the actual success rate to 98.4%.

Figure 4-7 shows the results for grasping all 10 objects in simulation with higher levels of initial uncertainty (standard deviations of 5 cm in x, 5 cm in y, and 30 degrees in θ). Note that at this level of uncertainty, it is possible for the object to be 15 cm and 45 degrees away from the initial estimated position; the object is somewhere on the table in front of the robot, but the robot has only a rough idea of where, and so it is essentially groping around blindly. With this much uncertainty, executing the goal-seeking grasp open-loop seldom succeeds. Using just RRG-Goal is sufficient for many of the objects, with the notable exception of: 1) the box (this is the example we gave when motivating information-gathering grasps; it cannot gather information about the location of the end of the box) and 2) the powerdrill, for which the goal-seeking grasp is grasping a nearly-cylindrical handle that gives it little information about the orientation of the drill (but where the orientation is important for triggering the drill, and for later use). When the target grasp is able to gather information about all relevant goal dimensions, RRG-Goal is generally sufficient.

Using RRG-Info brings our success rate above 95% for all objects except the cup and teabox. The small sizes of both the cup and the teabox cause the fairly coarse grid sampling (the grid spacing that we use is 1 cm and .1 radians for this level of uncertainty) to be too poor an approximation for the actual continuous observation probability distribution. At a resolution this coarse, the majority of the probability mass sometimes lies between the grid points, and so the grid point with the highest sampled probability may not actually be the one closest to the actual most likely state. This problem is shown in Figure 4-8, which shows the observation probabilities for the grasp shown in Figure 4-9. The red circles show the probabilities on a finer grid sampling (.25 cm, which is that used for the 1 cm/3 deg level of uncertainty) overlaid over our 1 cm grid.

Thus, localizing the object within a 1 cm goal region is not always possible with our fixed grid resolution. However, for both the cup and teabox, the planner is always able to localize the object to within a small area, and moreover, knows that it has done so. If we used a variable-resolution grid that switched to the same grid used at the low uncertainty levels when the planner became sure that the belief state is contained within the smaller grid, we would most likely obtain results similar to those in the low-uncertainty graph.

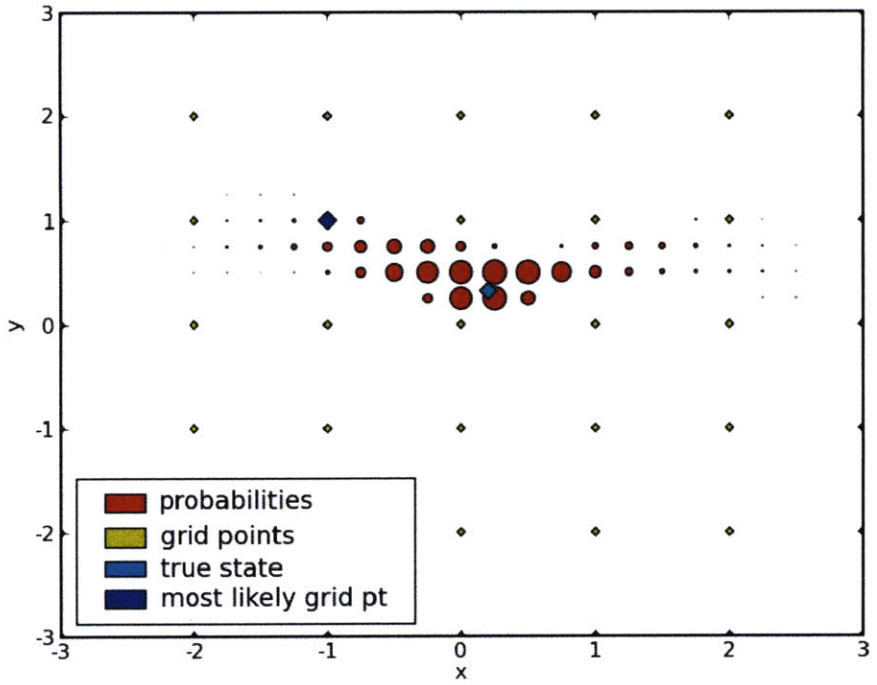

Figure 4-8: Observation probabilities for a cup grasp: most of the probability mass can lie between grid points, causing a poor function approximation and an incorrect most likely state choice.

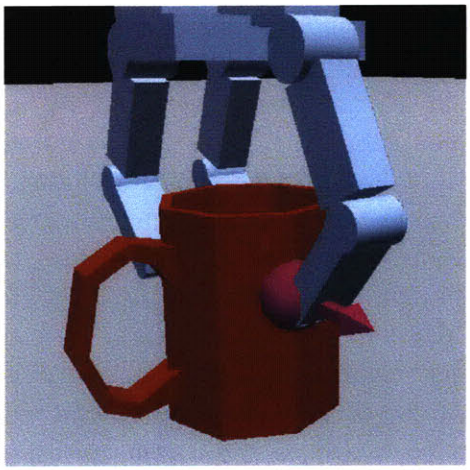

Figure 4-9: The cup grasp referenced in Figure 4-8.

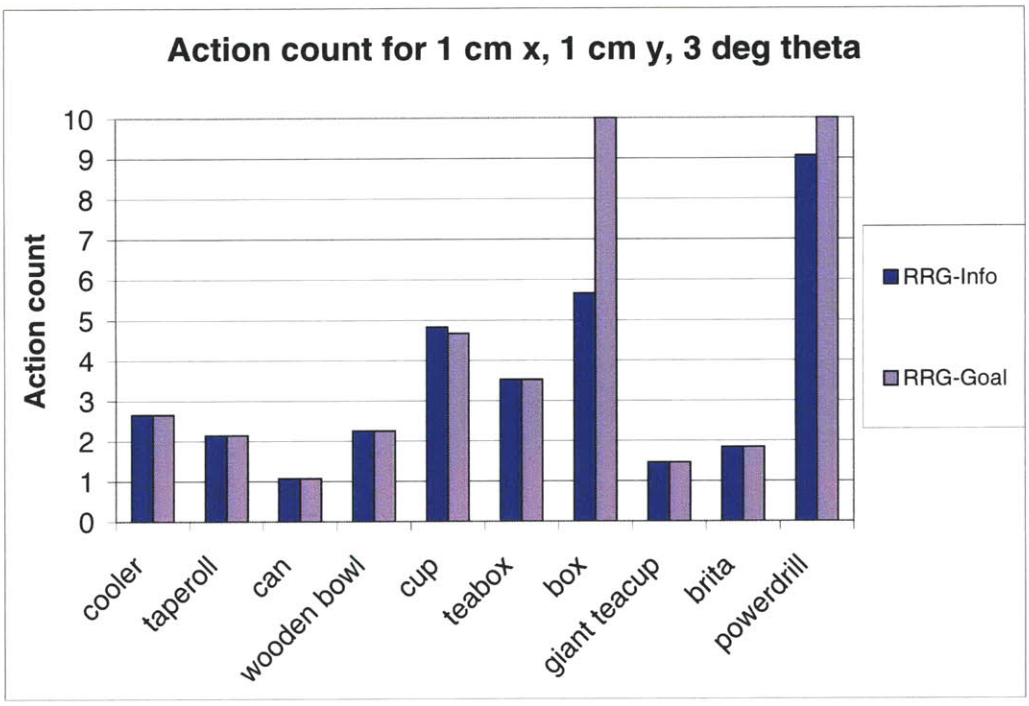

Figure 4-10: Action counts for all objects at low uncertainty (1 cm x, 1 cm y, 3 deg θ).

Figures 4-11 and 4-10 show the action counts for the same experiments. The can very seldom takes more than one action at the low starting uncertainty levels, since the can goal-seeking grasp is an enveloping grasp that almost always succeeds on the first grasp. The cooler, taperoll, wooden bowl, giant teacup, and Brita take approximately two actions at low uncertainty: one to locate the object precisely and the second to move to the correct location, grasp, and declare success. The cup, teabox, and box require a few more actions to locate within their more narrow goal regions. At high uncertainty levels, all objects take more actions while the robot gropes blindly around the workspace looking for the object.

RRG-Goal cannot collect enough information using just the goal-seeking grasp to be certain that the grasp is within the goal region for both the box and the powerdrill, and so it always continues to execute the goal-seeking grasp until it runs out of actions, for either start uncertainty. The same holds for the cooler at high uncertainty; at low uncertainty the goal region is large enough that the uninformative nature of the goal-seeking grasp does not matter. At high uncertainty levels, RRG-Info also requires a large number of actions to grasp the cooler, because both information-gathering grasps are often out of reach, so once the cooler is approximately localized, it often no longer has access to anything but the goal-seeking grasp.

The powerdrill, due to its narrow goal region, takes about 9 actions to localize for certain within the goal region with low starting uncertainty. On the other hand, it only takes about 8 actions at high uncertainty, despite requiring up to 4 actions

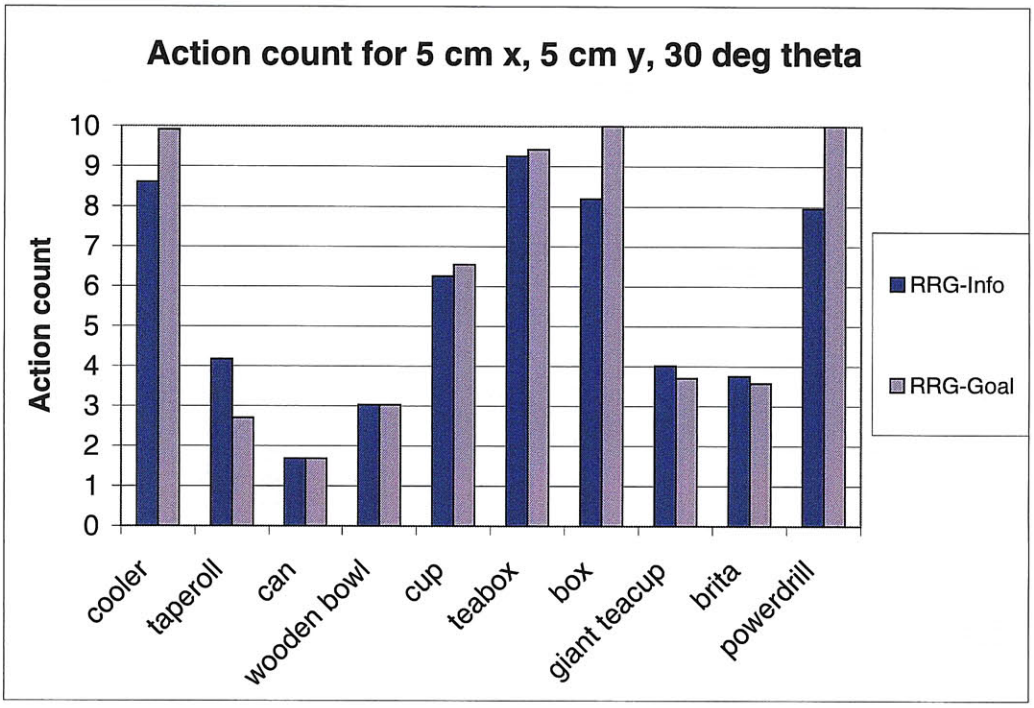

Figure 4-11: Action counts for all objects at high uncertainty (1 cm x, 1 cm y, 3 deg θ).

to locate to within the low starting uncertainty levels. This is because the coarse grid resolution makes it slightly too confident at times, which also results in a lower success rate, for similar reasons to the cup example in Figure 4-8.

4.2.1 Comparisons of various algorithms with the powerdrill

Figure 4-12 shows parametric results for just the powerdrill in simulation at the 5 cm/30 degree level of uncertainty, where δ was varied to show the trade-off between the number of actions executed and the probability of success. The five strategies used here are RRG-Goal, RRG-Info with a horizon of 3, 2, and 1, and RRG-Info with a horizon of 1 and entropy of the belief state as a static evaluation function instead of probability of success. Each point on the graph represents the average number of actions taken before termination and the percent successful for more than 100 simulated runs. Just executing the goal-seeking grasp repeatedly does not work well at all for this object, whereas just using a horizon of 1 works reasonably well. Increasing the horizon to 2 causes the planner to choose actions that result in a lower probability of success after just 2 actions, but that pay off in terms of a higher probability of success for fewer actions later on.

The greatest advantage of searching deeper in the search tree occurs when different information-gathering actions act as 'funnels' for one another. The powerdrill grasps (shown in 4-3) are a good example. Infograsp 1 is likely to provide information about

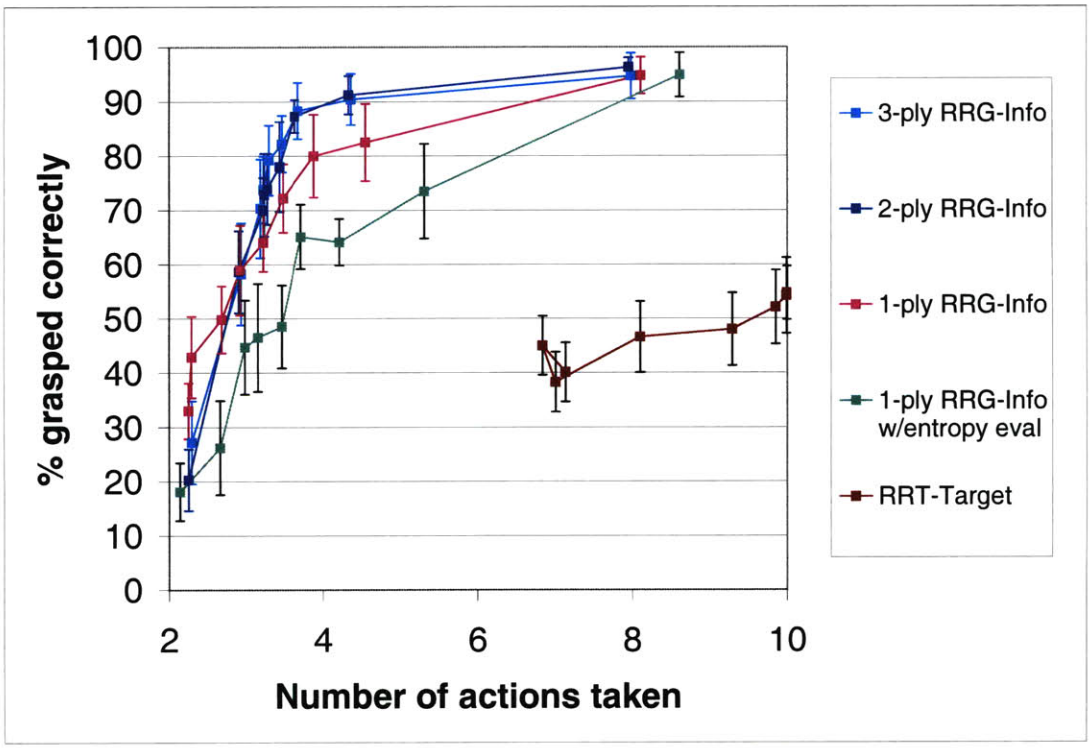

Figure 4-12: Comparisons of various algorithms (RRG-Info with a horizon of 3, 2, and 1, RRG-Info with a horizon of 1 and entropy as a static evaluation function, and RRG-Goal) with the actual robot. Error bars show estimated ranges for one standard deviation.

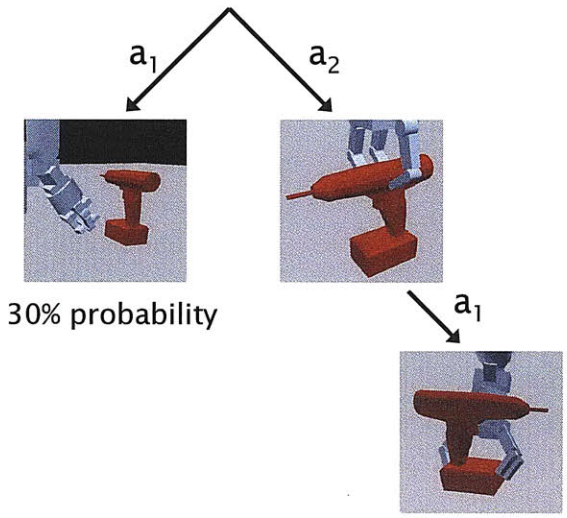

Figure 4-13: A tree showing the outcomes of two powerdrill actions, for a planner with a horizon of 2.

all three dimensions of uncertainty (x, y, and θ) if it succeeds, whereas infograsp 2 only provides information about x and θ if it succeeds. Thus, from the initial belief state with uncertainty levels of 5 cm in x, 5 cm in y, and .5 radians in θ, if the planner only searches with a horizon of 1, it will choose infograsp 1, because it needs information about all three dimensions in order to have any hope of succeeding. However, infograsp 1 also has a fairly high chance of missing the object entirely. Searching with a horizon of 2, the planner will notice that infograsp 2 acts as a 'funnel' for infograsp 1–it is much less likely to miss the object, and once the object is located in x and θ, infograsp 1 succeeds in gathering useful contact information with high probability. These outcomes are shown in Figure 4-13. Thus, the belief state is narrowed down faster by searching with a horizon of 2 instead of a horizon of 1. However, searching beyond 2 deep yields no additional benefit with this set of WRTs, and in general is unlikely to provide sufficient benefit to justify the significant additional cost of searching to a deeper horizon.

Although even planning with a horizon of 1 succeeds on the powerdrill with essentially the same probability as using a horizon of 2 or 3 after 10 actions, searching 2 or 3 deep reduces the number of actions needed to succeed more than 90% of the time from an average of 7 actions to an average of 4 actions, which is a dramatic speedup. Because even a horizon of 1 is usually sufficient to pick reasonable actions, if we have a large number of WRTs to select from (as we might need to have in the presence of a great deal of clutter), we can still select reasonable actions quickly.

With the exception of the reorient actions, our action selection problem might look, at first glance, to be submodular in nature. Submodularity is a property possessed by many problems that involve purely information-gathering actions. It refers to the property of having diminishing returns: if A and A' are sets of actions already completed, and $A \subset A'$, then for any additional action X, $F(A \cup X) - F(A) \geq$

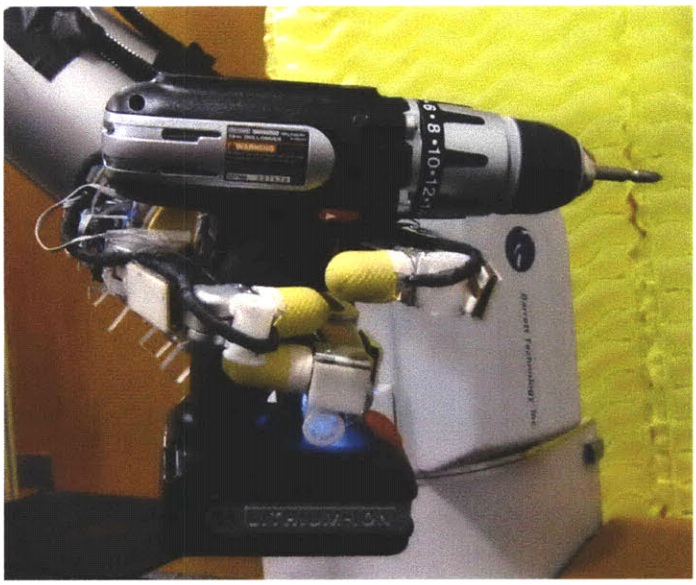

Figure 4-14: The robot grasping the powerdrill.

$F(A' \cup X) - F(A')$, where F is a function such as the entropy of the belief state [25]. In other words, adding a new action on top of a smaller set of actions gathers more information than adding it to a larger set of actions. If our problem were submodular in nature, as many information-gathering problems are, then using a modified greedy algorithm would be guaranteed to do at least $\approx$63% as well as selecting optimal actions. However, our problem is not submodular. First of all, this is because our actions are not purely information-gathering; the object moves when pushed, and so entropy can actually increase after an action. However, even if we assumed that the object were cemented in place and could not move, the actions that are available at each time step depend on the current belief state, and so the order of actions taken matters, as shown by the funneling powerdrill infograsps. If we considered a much larger, fixed set of robot motions, rather than parameterizing them based on the current belief state, then our problem would be submodular. This would make the action branching factor far too large to search in a reasonable amount of time, however, and so we cannot make our problem submodular by doing so. We parameterize our actions according to the most likely state in the hope that the set of motions that we consider are among the best actions to take next, and so while our problem is not actually submodular for the reasons described, it is close enough in nature that it is not surprising that a greedy algorithm does so well, and that searching more than 2 deep is not useful.

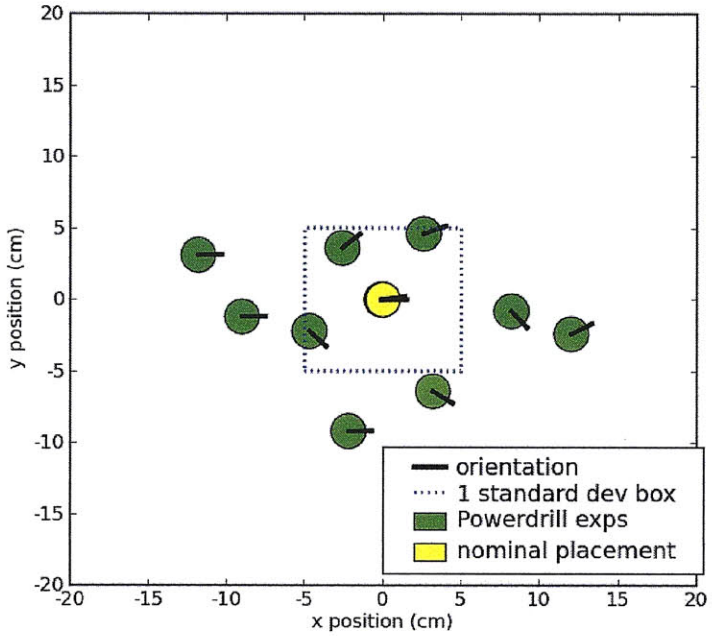

Figure 4-15: Locations of the 10 randomly-generated powerdrill grasping experiments.

Figure 4-16: The robot grasping the Brita pitcher.

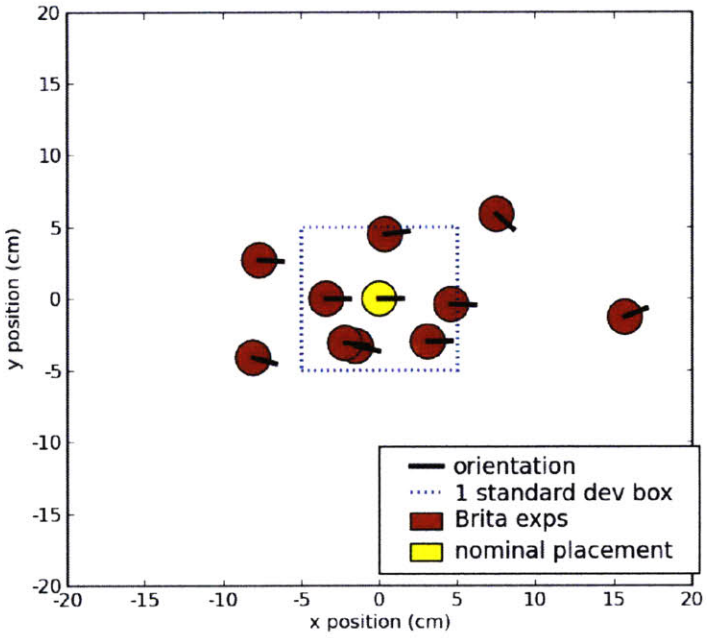

Figure 4-17: Locations of the 10 randomly-generated Brita grasping experiments.

4.3 Real robot results

4.3.1 Powerdrill and Brita

On the real robot, we ran 10 experiments each for both the Brita pitcher and the powerdrill at initial uncertainty levels of 5 cm in x, 5 cm in y, and 30 degrees in θ, again with random positions drawn from the same distribution. The object locations for the Brita experiments are shown in Figure 4-17, and the object locations for the Powerdrill experiments are shown in Figure 4-15. Both objects were grasped stably and lifted successfully 10 out of 10 times, with the trigger being pressed successfully on the powerdrill and the Brita pitcher being grasped properly by the handle.

4.3.2 Other objects

For the other objects, we ran five experiments each: 1 at uncertainty levels of 1 cm/3 deg with RRG-Info with a horizon of 2, and four at uncertainty levels of 5 cm/30 deg (three with RRG-Info with a horizon of 2, and one with just RRG-Goal). The object placements are shown in Figures 4-19, 4-20, and 4-21, and the results are shown in the chart in Figure 4-22.

Most of the grasps succeeded. Two grasps (the box and teabox with just RRG-Goal) failed as expected, due to the goal-seeking grasp's inability to collect information in a relevant goal dimension. However, when we say that they "failed", we mean

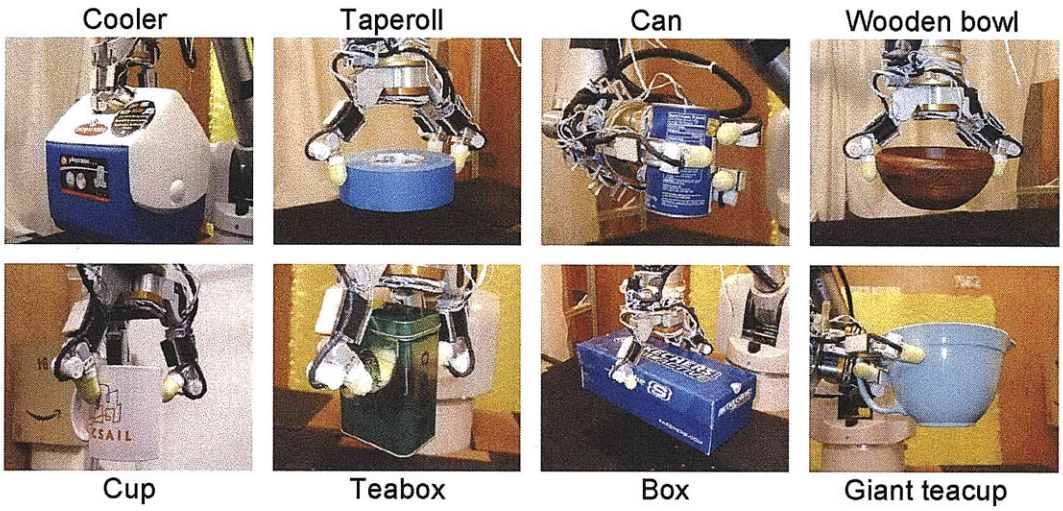

Figure 4-18: The robot grasping the other eight objects.

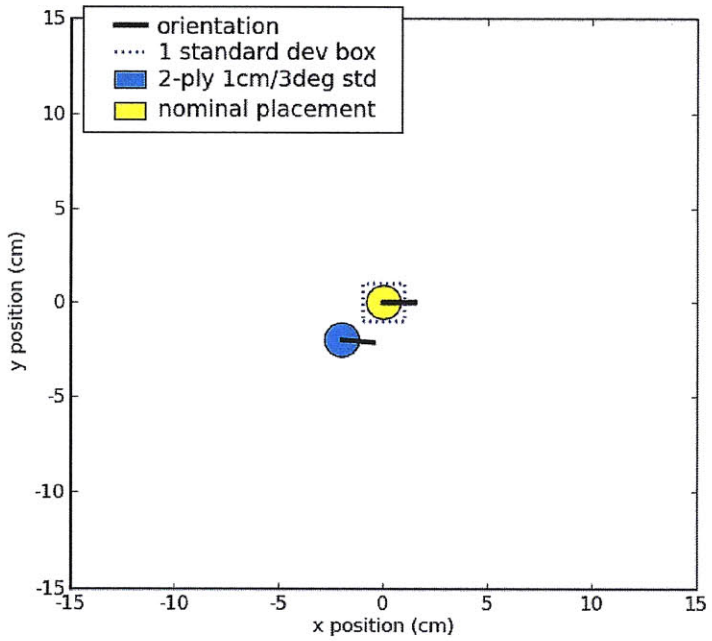

Figure 4-19: Location of the RRG-Info, low-uncertainty experiments on the other eight objects.

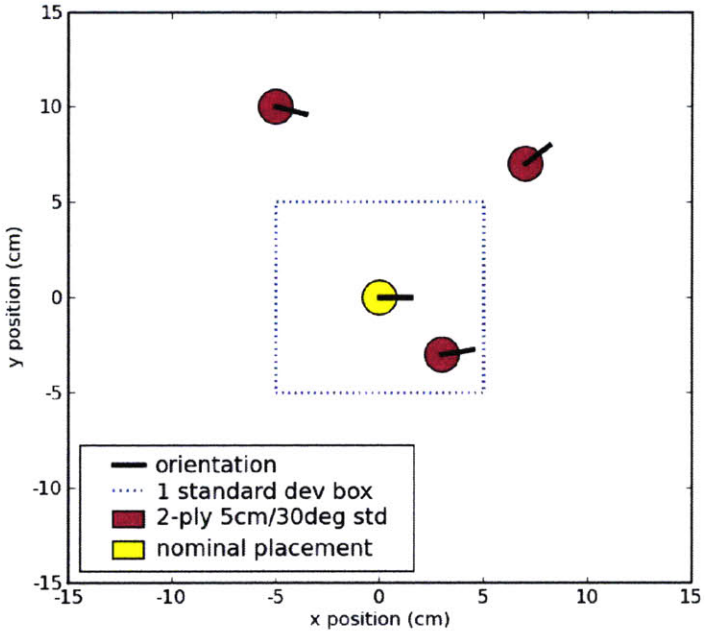

Figure 4-20: Locations of the RRG-Info, high-uncertainty experiments on the other eight objects.

that they did not execute the grasp that we asked for, as expressed by the goal region. In both cases the robot managed to pick the object up and off the table, which for many grasp planners would be deemed a success.

The cooler grasp that failed was due to executing an information-gathering grasp that swept horizontally across the workspace but hit the corner of the cooler on a part of the hand with no sensors, thus shoving the object out of the workspace. The can was knocked over by a jerky hand movement. The cup and teabox failures were due to the same failure mode discussed in the simulation results; the teabox in real life adds the additional complication of being too light to sense without shoving the object significantly.

The giant teacup grasp failed because the mesh (especially at the handle) is very thin, and our approximation of the robot's swept path can miss the fact that the fingers go through the handle in its most likely location. This is because our "swept path" approximation is actually a sampled set of robot locations along its desired path, not a continuous swept volume. Thus, it is possible for two samples of the finger location to be on either side of a thin object such as a handle, with no collisions in either sample. If this happens, the probability of the swept path will be the same as if there were no object in the way. Also, with a thin object, even if the finger samples end up in the middle of the handle, the depth of collision is low enough that it only lowers the estimated probability of the most likely state slightly, since the probability is based only on the depth of collision, as described in section 3.8.3. If that happens,

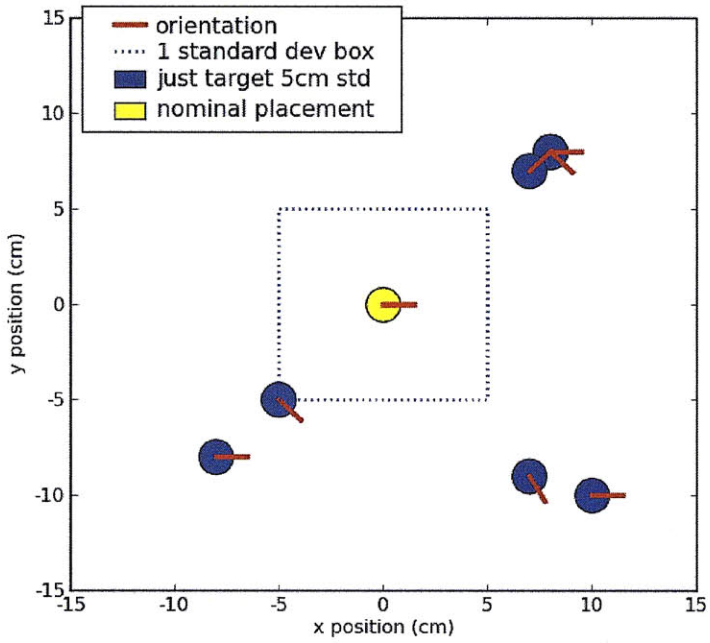

Figure 4-21: Locations of the RRG-Goal, high-uncertainty experiments on the other eight objects.

= succeeded = failed = failed as expected	cooler	taperoll	can	wooden bowl	cup	teabox	box	giant teacup
1 cm/3 deg RRG-Info (-2x, -2y, -5 deg)								
5 cm/30 deg RRG-Info (3x, -3y, 10 deg)								
5 cm/30 deg RRG-Info (7x, 7y, 40 deg)								
5 cm/30 deg RRG-Info (-5x, 10y, -15 deg)								
5 cm/30 deg RRG-Goal (≥5 cm, 45 deg mag)								

Figure 4-22: Results for grasping the other eight objects with the actual robot.

the most likely state often does not change, or changes very little, so the next action is often the same as the last. Each successive repeated action that sweeps out a path that goes through the most likely position of the handle lowers the probability of the most likely state slightly. However, it can take more actions than are available to rule out the current most likely state and search for the handle elsewhere. This failure mode is also seen in simulation.

Videos of the real robot experiments can be seen at
http://people.csail.mit.edu/kjhsiao/wrtpomdps.

4.3.3 Discussion

Several of the failures were due to limitations in the current implementation, such as using a fixed-resolution as opposed to a variable-resolution grid, having jerky robot control, using an imprecise swept path approximation, and not computing and taking into account the probabilities of WRTs failing due to contacts with sensorless parts of the hand. Nonetheless, there are several general principles that we can observe from our experiments. The first is that a small search horizon is effective in our framework; a horizon of 1 is usually sufficient, and a horizon greater than 2 is generally not useful, which means that planning does not have to be very expensive. The second is that we are still able to choose effective actions despite the aggressive observation clustering that we do to limit the observation branching factor. The third is that one of the limitations of our framework is the quality of the transition model. Our system currently works well for objects that either do not move much when bumped into during a guarded move, or else that are large enough, or have sufficiently large goal regions, that information-gathering grasps (often on surfaces far away from the center) can provide enough information to overcome the fact that the object moves significantly every time we touch it. If we had a more predictive transition model that could more accurately estimate how objects move when we bump into them, we could further improve our results. A more accurate transition model could even enable us to add actions that purposely push objects in order to gain information, by, for instance, shoving them against walls or other immovable objects, as in [1]. Finally, some objects are too light or fragile to bump into even with the most sensitive touch sensors, and this method is unlikely to work well for those objects; in many of those cases, it may be better to use "pre-touch" sensors, such as IR or electric field sensors, to localize objects without having to touch them [35, 23].

4.4 Computation times

Most of our code is non-optimized Python, which makes our computation much slower than it ought to be. Nonetheless, we present these computation times to give one a sense of where the bottlenecks in our system currently lie; we predict that a better implementation could reduce all of the following items to requiring only a few seconds.

Searching for the first action for the powerdrill with five available WRT actions takes about 3 seconds to search at a horizon of 1, 30 seconds to search at a horizon

of 2, and 180 seconds to search at a horizon of 3. The available grasps are shown in Figure 4-3; infograsp 2 is considered both as a normal information-gathering grasp and also as a reorient grasp. Fortunately, the first action can be stored for re-use each time we wish to grasp the same object with the same set of actions, and the search time goes down as the belief state narrows; the second action takes 71, 21, and 2 seconds, and the third action only 56, 17, and 2 seconds, for 3-ply, 2-ply, and 1-ply respectively. Again, with an optimized implementation, this could take much less time.

Another aspect of the algorithm that takes significant time at each time step is the current belief update, which includes the action update, the calculation of observation likelihoods for the actual observation seen, and the observation update. This takes approximately 30 seconds on average for all objects at high uncertainty levels, much of which is spent doing the swept path calculation; with a better swept path calculation, this could also take much less time.

The other major time-consuming part of the algorithm in the implementation used in the experiments is the calculation of feasibility for all candidate WRTs, which includes trying to find consistent waypoints and reasonable joint angles for barely-out-of-reach waypoint locations (as described in Appendix B). When all WRTs are feasible for all object positions, this takes only a fraction of a second. However, when the WRTs can become infeasible, as with the goal-seeking grasps for the powerdrill, Brita, and giant teacup, and with many of the information-gathering WRTs for the other objects, this calculation can take as much as 20 seconds on average due to fixing inconsistent and barely-infeasible waypoints, as explained in Appendix B. We have since implemented a version of these components that can calculate feasibility and determine consistent joint angles for waypoints in only a fraction of a second, so this part of our algorithm need not take so long.

4.5 Chapter summary

In this chapter, we presented results for using WRT POMDPs both in simulation and on the real robot. Our results demonstrated that the WRT-Goal algorithm alone works well on many objects but is insufficient in some cases, and that the WRT-Info algorithm usually succeeds where WRT-Goal is insufficient. We also compared the performance of WRT-Info while grasping a powerdrill with different search horizons, and showed that the search horizon need not be more than 1, and does not do better with more than 2. This property makes using WRT-Info quite tractable.

Chapter 5

Conclusions

Our objective in this thesis was to design a general approach to the problem of robustly grasping objects in particular goal locations, taking actions to gather information as needed to ensure robustness, without having to be overly cautious. We assumed that we had reasonably accurate models of our objects, but that those objects could have significant pose uncertainty. We also assumed that we had reasonably accurate robot control, with known proprioception error, and a known sensor model.

Our general approach was to use POMDPs to model the problem, which allowed us to reason explicitly about the uncertainty in the world, and to select appropriate actions based on the current belief state. In doing so, we chose to spend more time modeling the world (robot, object, sensors), in exchange for spending less time and human effort in developing object-specific policies that might add similar levels of robustness to our grasps.

We have presented two approaches that use POMDP models to solve this problem, one using finite-space POMDPs and off-line policy generation, and one using continuous-space POMDPs and forward search for action selection. These approaches were used in two different domains: 2-D grasping of rectilinear objects, and 3-D grasping of mesh objects. Both approaches had to deal with the same challenges to using POMDPs to model manipulation problems, which are that manipulation is a domain that naturally has continuous state, action, and observation spaces, and computationally complex observation and transition models. In both cases, we managed to reduce our belief state to a compact representation that captures just the uncertain parts of our belief. We used guarded compliant motions as our actions in both approaches, because of their ability to act as "funnels" that produce configurations with multiple contacts between the robot and an object for large ranges of states, enabling us to gather information efficiently. Finally, in both methods, we used geometry-only simulations to compute and store our observation and transition models in a way that allowed us to avoid having to re-compute them online.

While the two approaches had very different sets of states, actions, and observations, the key to many of our choices was to express states, actions, and observations as being relative to the object pose whenever possible. In the finite-space POMDP model, the abstract states were blocks of hand poses relative to the object. In the continuous-space POMDP model, our actions were parameterized by the most likely

object pose. Also, in computing our observation model for the continuous-space POMDPs, we only had to compute a single set of observations relative to a single estimated object pose, and could then re-use it for other estimated object poses. Expressing things relative to the object pose makes the absolute pose of the object unimportant (excluding considerations of reachability and obstructions due to obstacles) while planning, which is crucial for keeping the complexity of our planning processes at a reasonable level.

5.1 Summary of Contributions

The contributions of this thesis are as follows:

We presented the first formulation of manipulation as a POMDP, in using a finite-space POMDP model for the simple but still moderately interesting domain of grasping rectilinear objects in 2D. Using guarded compliant motions as our action set allows us to use model minimization techniques to create abstract states and transitions, and the resulting abstract POMDP models are compact enough to solve with an offline, approximate optimal policy solver. We showed that the resulting policies are very robust, both in simulation and on a real robot, and that they allow us to grasp robustly even in the face of: observations that both use extremely limited information and are extremely unreliable; fairly significant shape uncertainty; and the transition model being significantly different than the actual nominal behavior of the robot.

We then extended the use of POMDPs to modeling arbitrary mesh objects in 3-D, with completely different state, action, and observation choices. We introduced the concept of World-Relative Trajectories (WRTs), which give us access to a large set of possible actions while only having to consider a small, discrete set of parameterized actions for a given belief state. We explained how to use POMDP forward search to select appropriate WRTs online in a tractable way by computing and clustering observations in our observation model offline, for re-use while planning online. Finally, we demonstrated the full framework both in simulation and on a real robot, and showed that planning with a small search horizon (no more than 2) is generally sufficient for selecting actions effectively.

5.2 Future Work

5.2.1 Implementation improvements to RRG-Info

There are many minor implementation improvements that we would like to do to improve the performance of RRG-Info, as mentioned in section 4.3.3. These include:

- Using a variable-resolution grid. The planner can shrink the size of the workspace represented and increase the resolution of the grid, when it detects that the current belief fits within a smaller workspace. To do so, we can use exactly the same framework, only computing and storing the observation model at several resolutions, and shifting among them as appropriate.

- Adding an additional function to the observation model, $R_\tau(w, e)$, which has value *fail* if a collision will occur that cannot be sensed (e.g., with a part of the hand with no sensor pads). Such a collision is likely to cause the object to be knocked over or moved significantly, and so we will consider it to be a failure. We will also consider it a failure if a contact would cause damage to the robot (e.g., by crushing delicate sensor cables). This function can be computed simply by marking some of hand points in the hand point model (described in Appendix C) as points that cause failure if contact would occur there. Using this function, we could predict the probability of an action resulting in catastrophic failure given the current belief state, and then use that probability in planning which action to take next.

- Improving the swept path calculation. Currently our swept path approximation does not distinguish between going through thin walls or brushing past the corner of a solid object, as described in section 3.8.3. If we switch to sampling a reasonable number of trajectories that are shifted versions of the nominal trajectory, we could use the ratio of non-contacting trajectories to the total number of trajectories sampled as our estimate of $P_{free}(w, \tau)$. This would be a more accurate estimate that could distinguish between the two cases.

5.2.2 Adding obstacles

In order to add consideration of obstacles to our framework, we need a laser rangefinder or some other method of estimating which areas of space are off-limit to the robot. We could then assume pessimistic boundaries for the obstacles and eliminate actions that would cause collisions from consideration at each time step. However, we could also track how certain we are that various regions of space are occupied, and observe obstacles by touching them. Actions that do so could be included in our forward search in terms of how likely they are to make our standard WRT actions available when they would otherwise be eliminated due to obstruction. Alternatively, if we could estimate the effects of trying to move obstacles, we could also include "clearing" actions in our set of available actions, which would likewise make obstructed WRT actions available in later steps. However, because actions of these sorts change the actual state of the world, it may be that we would have to search deeper in our forward search tree in order to make effective use of them.

5.2.3 Pushing actions

In a similar vein, if we could accurately predict the effects of pushing our target object, we could add actions that attempt to push the object against other surfaces. Doing so could enable us to gather information about the object pose by constraining the possible poses it could end up in after being pushed, as in [1]. As long as we have a proper observation and transition model for a new action, it can be included in our framework with minimal changes.

5.2.4 Other sensors

Other sensors such as laser rangefinders, vision, or "pre-touch" sensors such as IR or electric field sensors, could also be added to our framework with minimal changes, as long as we can estimate the relative probabilities of various object poses given the observation. Using IR or electric field sensors at the fingertips could allow us to gather information without disturbing the object; using a laser rangefinder or ongoing visual tracking could allow us to detect and respond to cases in which we have moved the object significantly due to touching it with part of the hand with no contact sensors, or in which we have actually knocked the object over.

5.2.5 Shape uncertainty

In order to deal with significant shape uncertainty, we could parameterize the shape of the object and add it to our belief state, and otherwise leave the rest of the framework unchanged. However, doing so might cause the belief state to be too high in dimension to plan in a tractable way. On the other hand, it might be sufficient in some cases to just keep updating a single estimate of the object shape based on our contact observations. We could assume that the shape is correct while computing our belief about the object pose, and then based on our belief state, change the shape of the object when our observations are inconsistent with the current object shape under any object pose with reasonable probability.

5.2.6 Increasing dimensionality

Our current grid-based framework could be difficult to extend to additional dimensions. For instance, if we have uncertainty about the z-coordinate of the object in addition to x, y, and θ, our belief state becomes 4-dimensional, which already makes belief state update and computation of our observation and transition models unwieldy. In order to retain a compact representation of the belief space while still being able to represent multinominal distributions, we could move to a belief state expressed as a mixture of Gaussians. It is still possible to do fast belief updates with such a representation, as explained in [8]. Having to pre-compute the observation and transition models in higher dimensions could be more problematic, however. For situations in which the robot is not groping around blindly on the table, but instead the object has a lower level of pose uncertainty that we still need to deal with robustly, it may still be possible to compute our models offline. Alternatively, we could just sample as many observations as we have time for online; as our belief state narrows, we would need fewer observation samples to cover the non-negligible states. More importantly, because most of our actions just collect information anyway, and because we are using receding horizon planning, it may be that using few observation samples would be akin to the drastic observation clustering and pruning that we already do, and thus the difference might be insignificant.

5.2.7 Task planning

Our goal conditions currently assume that the size of the goal region is given. However, our premise is that the size of the goal region is based on the range of positions that allow one to perform some task. The task could be simple and not involve other objects, as when grasping the object in a particular way so that one could press a trigger. It could also be more complex, such as when doing multi-step assembly, or even just when trying to place the object in a particular location with constraints, such as in a cabinet or on a rack. We would like to be able to automatically figure out the required goal region on the object for grasping in order to perform such tasks.

5.2.8 Convergence properties

We would like to know if and when we could expect convergence when running RRG-Info, i.e., if we ran all of our grasps an infinite number of times, whether it could be guaranteed that we would know exactly where the object is, and if so, under what conditions.

5.3 Conclusion

Using POMDPs to model the problem of manipulation in the presence of uncertainty allows us to grasp objects robustly by automatically selecting appropriate actions to gather information and attempt to achieve our goal grasps. By reasoning explicitly about the uncertainty in the current belief state, we can estimate how likely we are to succeed in our grasp, and avoid performing unnecessary actions.

Using guarded compliant motions and abstract states to express the problem of manipulation as a finite-space POMDP that can be solved through off-line POMDP solvers allows us to generate highly robust policies. However, the domains in which this can be done tractably are fairly limited, and using off-line POMDP solvers prevents us from using a large amount of available information that could be used as observations if we were not extremely limited in our observation space.

Using WRT POMDPs and selecting actions through POMDP forward search allows us to to use continuous action and observation spaces, and to thus generalize to a wider range of problems, including grasping mesh objects in 3D. This framework can be used in any situation where we can provide a set of feasible trajectories to use as actions, reasonable models of the object, robot, sensors, and the effects of actions, and an uncertain belief space that can be represented compactly and updated quickly. These elements are sufficient for using the basic RRG-Goal algorithm, which already adds a great deal of robustness in many situations. In order to select actions in a tractable way through POMDP forward search (RRG-Info), we also require that the observation and transition models be computable off-line in a fashion that allows us to re-use them quickly on-line while constructing the forward search tree.

There are a number of conditions under which WRT POMDPs will fail:

1. If the space is so cluttered or constrained that any of our pre-computed trajectories would be infeasible if the object were in any pose but the one for which the trajectories were computed, or at least in many of the possible object poses, then our WRT actions would be useless.

2. If the available actions cannot gather more information when executed than the amount of entropy they add to the belief state, or if all actions have too high a probability of causing catastrophic failure (such as knocking the object over or shoving it out of the workspace), then we cannot make progress towards our belief-state goal. Both of these conditions are true, for instance, when using tactile sensors with an extremely light or crushable object.

3. We assume rigid objects, so significant changes would have to be made to deal with deformable objects.

4. Finally, the tractability of doing POMDP forward search depends on being able to pick reasonable actions that make progress towards the belief-state goal in a reasonable amount of time. In our case, we accomplished this by having a low search horizon, which was sufficient for our largely information-gathering actions and in the domains that we examined. If the actions are chosen such that the goal horizon is long, one would have to balance that with having a far lower branching factor on both actions and observations.

Despite these limitations, the WRT POMDP framework works for a wide variety of situations. It adapts automatically to wide ranges of uncertainty, gathering information and taking actions to achieve the goal as necessary, and also generalizes easily to using different or additional sensors or actions, as long as the above conditions are met.

Appendix A

Sensors

A.1 FSR sensor design

The sensors used in Chapter 2 are pressure sensors that can provide information about both the force and location of contact along a single axis.

Figure A-1 is a diagram of the sensors used for the outside and inside of a sensing finger. These sensors, designed by Ross Glashan, consist of two force-sensing resistor (FSR) sensors sandwiched between two thin aluminum plates, with two tiny squares of foam focusing the force onto the FSR sensor pads. This design ensures that a contact force anywhere on the surface of the aluminum plate is detected despite the tiny surface area of the FSR pads. In addition, since the contact force applied to the plate is detected at each sensor in proportion to the distance between the applied force and that sensor's pad, one can use the two sensors' force information to estimate the location of the applied force along the length of the sensor. Construction is simple and requires only cutting two small rectangles of aluminum, taping the FSR sensors to the bottom plate, soldering wires to the FSR leads, sticking small squares of sticky-backed cellular urethane foam onto the sensor pads, and finally loosely attaching the top plate with more tape. Our sensors are also covered with cellular urethane foam to add both higher friction and compliance. The FSRs are simply variable resistors, and so the simplest way to get readings from them is to make each FSR one of the resistors in a voltage divider, with the output voltage being sensed through an ADC.

Unlike the inner and outer sensors, the tip sensor has a tiny surface area and thus only has space for one FSR, preventing any sort of contact location estimate. For the larger inner and outer sensors, Figure A-2 is a graph of the estimated vs. actual position detected when a pointlike contact is applied to the layer of foam covering the sensor. Figure A-3 is a graph of the total force detected at both sensors for varying applied forces. The dotted blue line shows the sensor response when the force is applied at the center of the sensor, the dashed red and green lines show the response at 1 cm on either side of the center lengthwise, and the solid magenta line shows the response at .5 cm from the center widthwise. While there are many sensors with similar or better capabilities on the market, another advantage of this design is its cost: approximately $12 in parts and about 10 minutes of assembly time.

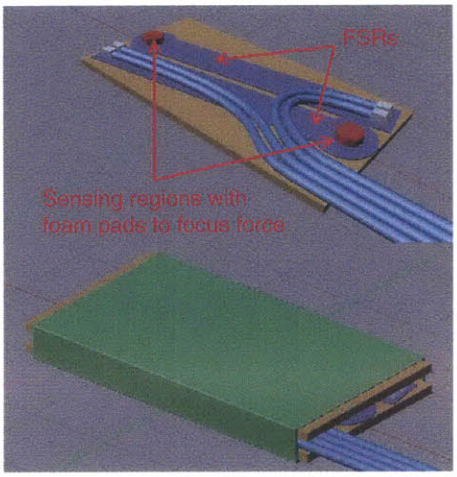

Figure A-1: Diagram of an FSR sensor that detects position as well as force of contact.

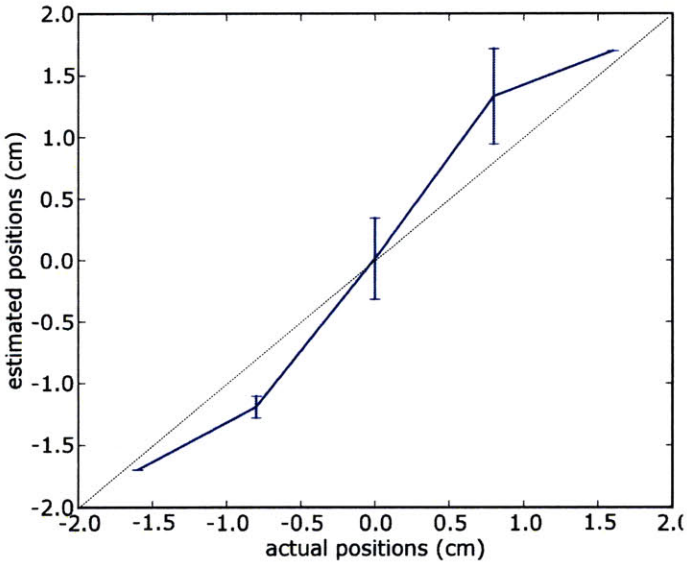

Figure A-2: Estimated vs. actual position of contact on sensor.

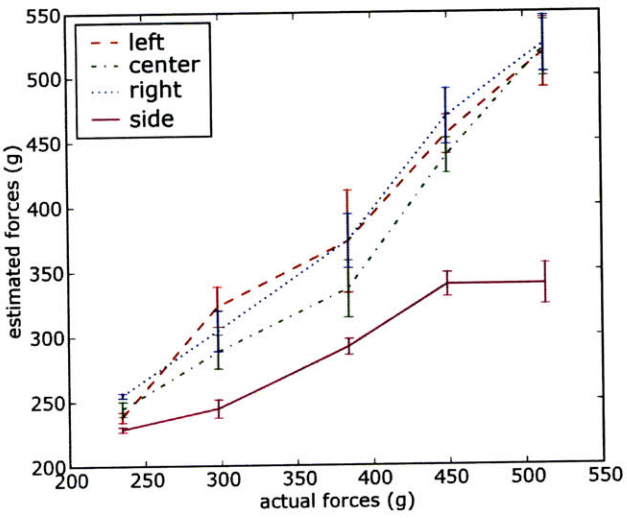

Figure A-3: Sum of estimated forces for varying forces applied at center of sensor, left and right length offsets of .8 cm, and .5 cm width offset.

Although these sensors are simple to make and have some fairly good force and location properties, they also have the unfortunate properties of requiring a fair amount of force be able to distinguish contact from noise, and also having quite a bit of fatigue. Repeatedly pressing the FSRs that we used results in requiring larger and larger amounts of force to generate the same decrease in resistance. This fatigue resets after leaving it alone for quite some time, but during a single grasp experiment, this can cause problems. Although the sensors would start out being fairly sensitive and reliable, after a few minutes, their reliability would drop precipitously. Fortunately, the point of the research that these sensors were used for was to be able to deal robustly with such noise. Our noise model for these sensors, which we only used to tell no-contact from contact in our experiments, was that contact would be missed 25% of the time, which was about as reliable as we could make them despite using fairly heavy objects that were made of solid aluminum.

A.2 Nano17 position and normal calculation

The fingertip sensors used in Chapter 4 consist of an aluminum shell attached to an ATI Nano17 6-axis force/torque sensor. The aluminum parts that we used were designed by Barrett Corp. especially for use with Nano17 fingertip sensors. The aluminum shell is cylindrical along its length, and spherical at the tip, for easier calculation of the contact location; fingertip sensors with this particular geometry were first presented in [6]. The Nano17 sensor itself is a short cylinder that sits underneath the base of the aluminum fingertip, as shown in part a) of Figure A-4. It attaches to the aluminum shell at the sensor attachment plane, and reports the

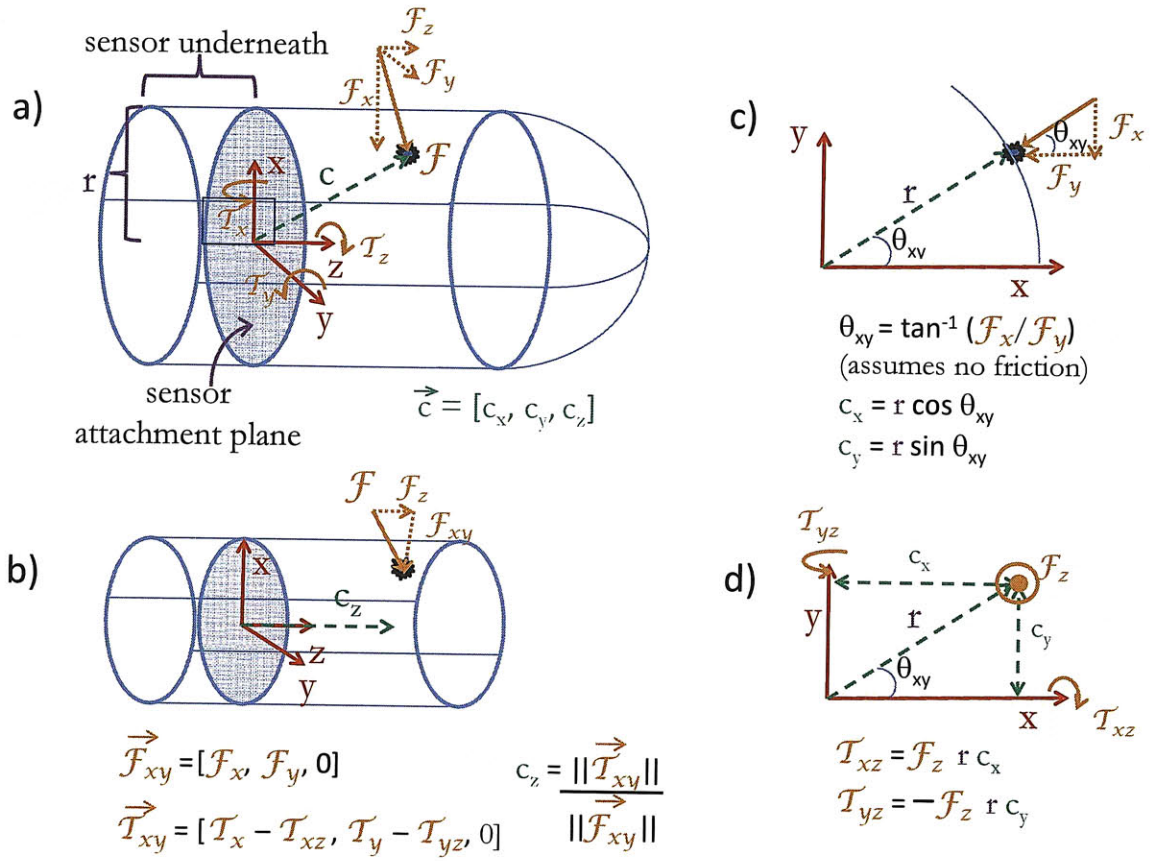

Figure A-4: Calculation of the contact position on a fingertip with a Nano17 6-axis force/torque sensor (for a single contact on the cylindrical side)

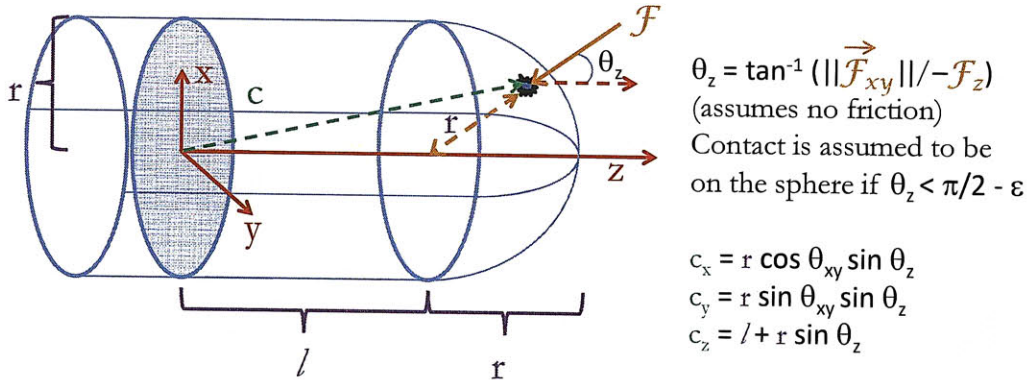

Figure A-5: Calculation of the contact position on a fingertip with a Nano17 6-axis force/torque sensor (for a single contact on the spherical tip)

forces and torques about the x, y, and z axes. In the figure, the three force values are denoted by F_x, F_y, and F_z, and the torque values by T_x, T_y, and T_z. In order to use this information in our framework, we need to convert those six values into $\vec{c}$, the position of the contact point on the finger, and $\vec{n}$, the contact normal. We assume that there is a single point contact; if there are multiple contacts on the finger they will be combined into a single contact point based on the net forces and torques seen by the sensor. When calculating $\vec{n}$ we assume the contact normal is orthogonal to the surface of the fingertip, which is simple to compute once we know $\vec{c}$. Computing $\vec{c}$ is slightly more complicated, and in our implementation there are two separate cases.[1]

Case 1: Contact on the spherical tip

When deciding whether the contact is on the cylindrical part or the spherical part of the fingertip, we compute θ_z, the angle between the net force F and the z-axis, assuming no friction, as shown in Figure A-5. If θ_z is less than $\pi/2 - \varepsilon$, where ε is a small, constant angle, we assume that the contact is on the spherical part; otherwise, we assume it is on the cylindrical part.

The assumption is that if there is a significant force in the $-z$ direction, it must be acting through the spherical part, since contacts on the cylinder would not generate significant z forces without friction. If there is a significant frictional force in the $-z$ direction, the contact can be misclassified, although the ε term is there to account for a tiny amount of friction. For a contact on the sphere, we assume that there is no friction and that F is orthogonal to the surface, and thus we can compute c_x, c_y, and c_z as shown in Figure A-5.

Case 2: Contact on the cylindrical side

If the contact is on the cylindrical part of the fingertip, we know that the forces are acting at a point that is at a distance r (the radius of the cylinder) away from the z-axis. When computing c_x and c_y, we again assume that there is no friction, and thus the force is orthogonal to the cylindrical surface. In that case, we can compute c_x and c_y as shown in part c) of Figure A-4.

When computing c_z, if we know that the contact is on the cylindrical part of the fingertip, we know how much force is due to friction, because frictional forces are parallel to the z-axis and thus have a magnitude of F_z. We can compute c_z based on $\vec{F_{xy}}$, the vector component of the net force with F_z removed, and $\vec{T_{xy}}$, the vector component of the net torque with the contribution from F_z removed, as shown in part b). To remove the contribution from F_z, we compute T_{xz}, the component of T_x due to F_z and T_{yz}, the component of T_y due to F_z, as shown in part d), and subtract them out to obtain T_{xy}, as shown in part b).

[1] An alternate method of computing the position of the contact, which we do not use in our implementation, is explained in Appendix C of [6].

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Robot Control

This appendix explains how a desired WRT action, τ, is executed on the robot. A WRT specifies a set of object-relative Cartesian hand poses that are waypoints along a desired Cartesian path. However, the robot is controlled via joint torques, and there are a number of steps required to convert from one to the other, as shown in Figure B-1. Each of these steps will be explained in detail in the following sections.

B.1 Cartesian paths

For a given WRT action τ and estimated object position e, we can convert the object-relative waypoints in τ into a set of global Cartesian hand poses, $\tau(e)$. Because e could be anywhere within the range of our start uncertainty, we would like to design τ such that $\tau(e)$ limits collisions with the object to only those that we have predicted while calculating $\Omega_\tau(w, e)$.

B.1.1 Avoiding unmodeled collisions

When computing $\Omega_\tau(w, e)$ for a given τ, we predict only those contacts that arise from the object-relative WRT start point on, not those contacts that might occur during the move from the global start position to the WRT start point. Thus, to avoid unmodeled collisions with the object that might occur while moving to the start point, we need to place the WRT start point well out of the range of where the object might be according to our start uncertainty levels. Figure B-2 shows what happens if we place the WRT start point too close to the object we wish to grasp (which in this case is a Brita pitcher). If the start of $\tau(e)$ can fall within the region where, given our start uncertainty levels, we might expect w to be, then we might see an unmodeled collision. As long as we make the move from the home pose to the start of $\tau(e)$ a guarded move, this is not disastrous for tracking the current belief state, since we can use an observation gained in this way like any other, but if our trajectories are likely to have unmodeled collisions, our planner might be choosing actions poorly.

Thus, our trajectories typically look like the first row of Figure B-3, where the

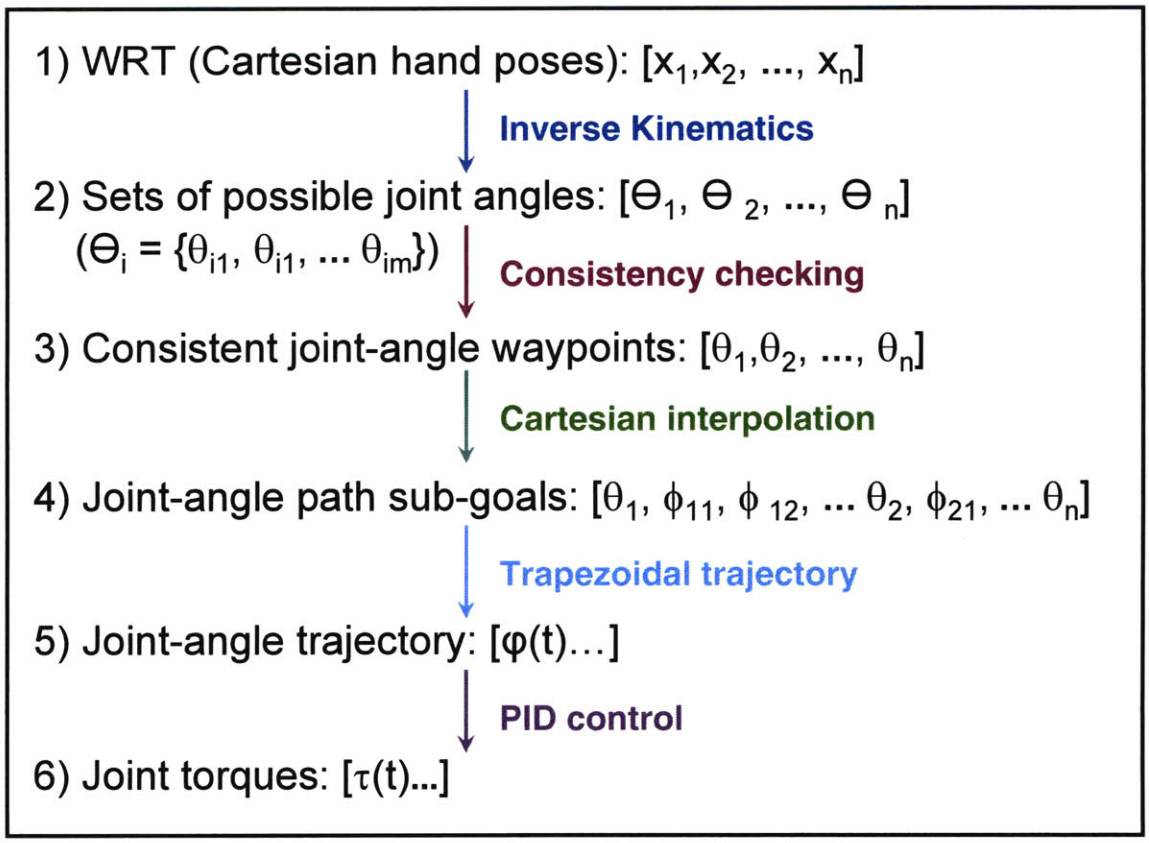

Figure B-1: The pipeline for converting WRTs into robot torque commands.

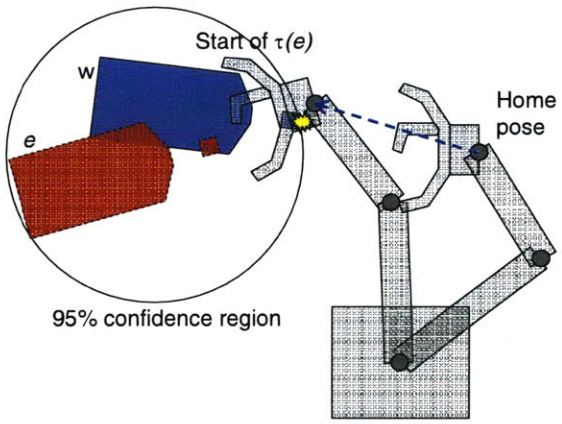

Figure B-2: Making the start point on a WRT too close to the object can result in unmodeled collisions.

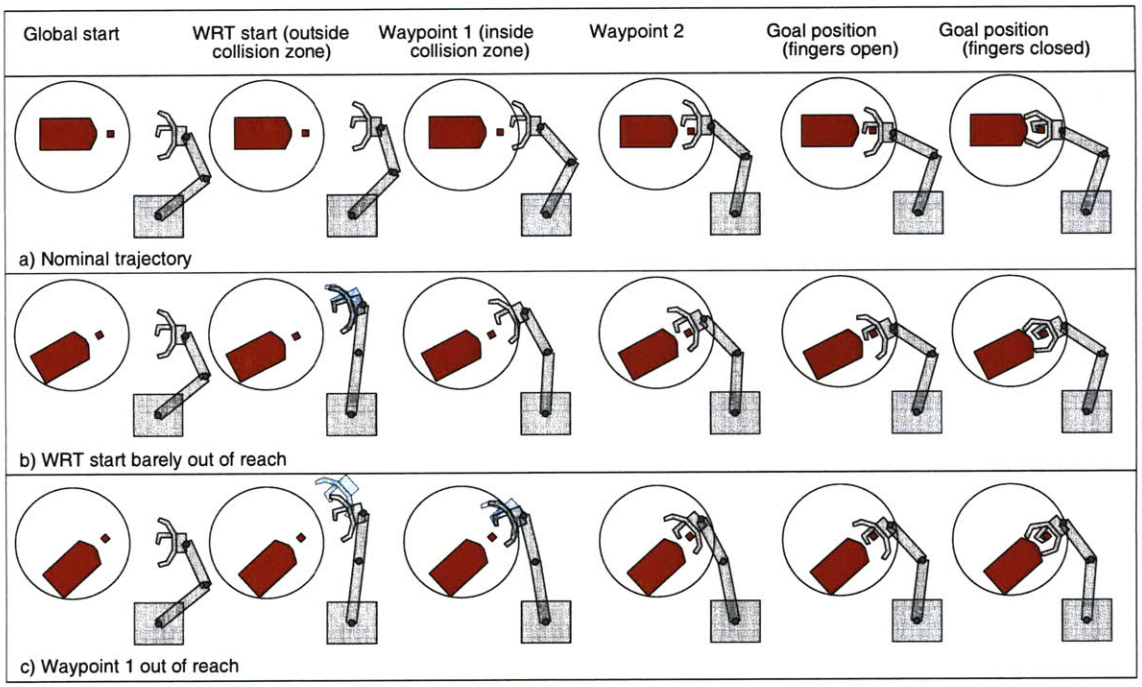

Figure B-3: Trajectories with different levels of feasibility.

robot starts at a fixed, global home pose, and then moves to the start of the WRT, which is placed far enough away from the object that for any w or e within the 95% confidence region, no collision will occur while moving there from the home pose.

We will denote the individual global, Cartesian hand poses in $\tau(e)$ as x_i, where i is an index from 1 to n, the number of waypoints in τ. Computing $[x_1...x_n]$ is the first step in our robot control pipeline, shown in Figure B-1.

B.2 Converting to joint-angle paths

Next, we need to be able to convert our Cartesian waypoints into a feasible path through joint-angle space, $[\theta_1...\theta_n]$. Such a path may or may not actually exist; if an appropriate path does not exist, $F(\tau(e))$, which expresses whether $\tau(e)$ is feasible, will have the value $false$, and we will remove τ from the set of available action choices in this planning time step. If τ is feasible, we need to be able to control the robot to move along the desired Cartesian path.

One fast way to control the robot hand to move along a set of Cartesian waypoints is to use a controller (such as PID) in Cartesian space directly, and to then translate desired Cartesian forces at each controller time step, $\mathcal{F}$, into desired joint torques, $\mathcal{T}$, via the Jacobian transpose:

$$\mathcal{T} = J^T \mathcal{F}$$

This is the method that we used in [22].

B.2.1 Path consistency

Using simple Jacobian transpose controllers can work well if a smooth joint angle path exists that could move the hand along the entire Cartesian path without approaching any singularities or exceeding any joint limits. However, it is often the case that even if inverse kinematic solutions exist at all points along a Cartesian path, there may be no consistent set of joint-angle waypoints.

Because our arm has 7 degrees of freedom, and because the hand pose only specifies 6 degrees of freedom, there is redundancy in the system. Each reachable hand pose can have a number of corresponding possible joint configurations; we will denote the set of possible joint configurations that make a desired x_i as Θ_i. However, due to joint limits, it is possible that an arm that is already at one joint configuration that puts the hand at the start pose cannot move to any of the possible end pose configurations along a Cartesian path.

For instance, it may be that all of the IK solutions for the endpoint require one wrist joint to be at an angle that is almost 2π radians away from the current joint angle, while the Cartesian path asks that the hand orientation remain fixed, and with the short path around the circle impossible due to joint limits. In that case, the hand cannot move from the start to the end smoothly along the Cartesian path without rotating that joint by nearly 2π somewhere in the middle. Such a path is shown in Figure B-4. In cases such as these, no incremental control algorithm can successfully move the hand along the desired Cartesian path. Instead, we would have to detect that no consistent path exists, attempt to find a joint-angle-compatible start configuration, and if possible move the arm to start there instead.

Also, even if we have feasible, consistent start and end configurations, the Cartesian path in between can still have infeasible points, or points that force the robot to pass near singularities. In these cases, using a simple Jacobian transpose controller to attempt to move from the start to the end can cause the robot to get stuck in a local minimum, or to oscillate or end up somewhere undesirable when passing near a singularity.

If we do not avoid trying to execute such trajectories, or find a way to execute them without failures of these sorts, we risk taking actions that gather no information, and thus failing to change the belief state at all. With an unchanged belief state, the planner will select the same problematic action again and again, and thus the grasp will be doomed to fail. In [22], a large number of failures in our simulation experiments were due to these problems, both while running the target grasp open-loop with perfect information about the object location, and while running the planner with WRTs. At high uncertainty levels of 5 cm in x, 5 cm in y, and 30 degrees in θ, running the target grasp open-loop with perfect information works about 70% of the time with the simple Jacobian transpose controllers. Using the trajectory planning techniques described in this section increases the success rate to about 80%. The target WRT is infeasible for the remaining 20% of object poses; reorientation is required to succeed in those cases.

In addition, when these sorts of problems are present, an incorrect e with a problematic $\tau(e)$ can cause failures, even if none of the τ would cause problems if the

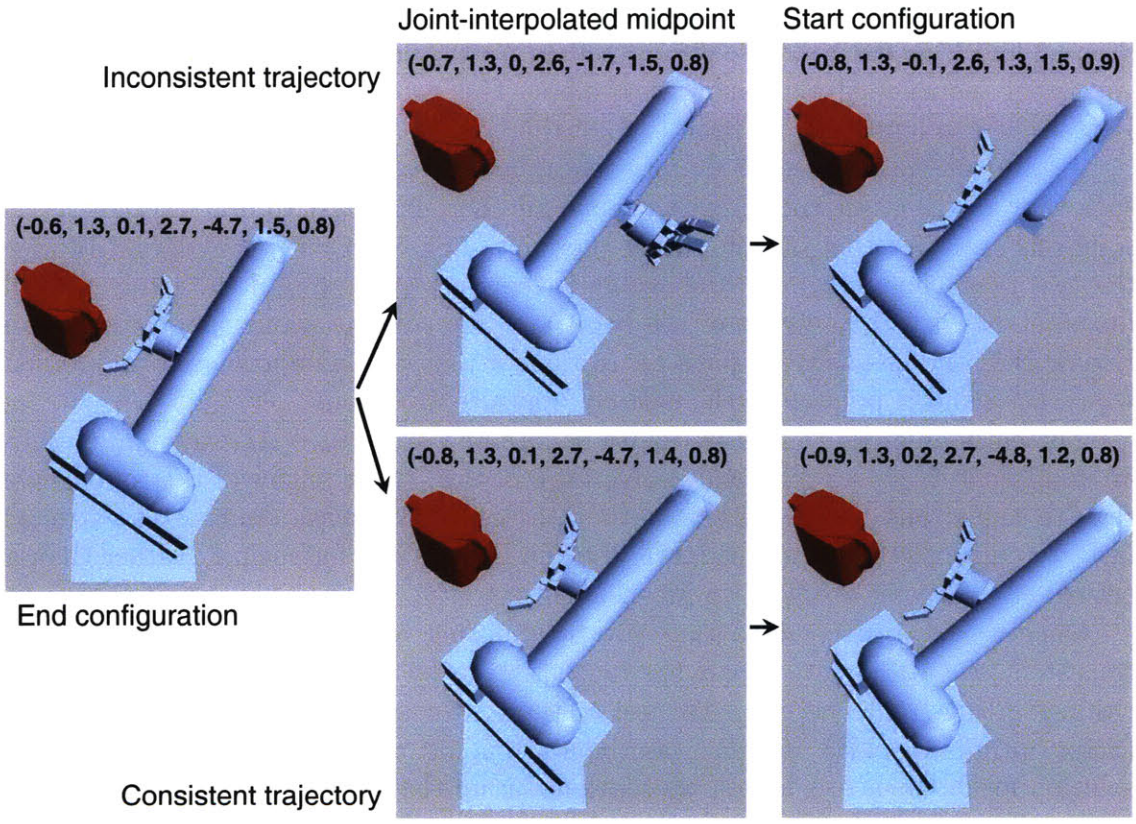

Figure B-4: IK can find inconsistent joint-angle waypoints.

robot knew the true w. Thus, figuring out roughly where the object actually is after fewer actions lowers the chances of failure due to problematic e. When grasping the Brita pitcher with the simple Jacobian transpose controllers, this means that at high uncertainty levels, using an entropy-based WRT planner with information-gathering actions has a much higher success rate (67%) than just doing robust re-execution of the target WRT (42%). With the trajectory planning techniques described in this section, we eliminate the extra failures due to the naïve controllers, and thus after 10 actions, both methods have the same success rate.

B.2.2 Finding consistent waypoints

Steps 2 and 3 in our robot control pipeline, shown in Figure B-1, involve finding a consistent sequence of joint-angle waypoints. This process involves selecting a single joint-angle configuration, θ_i, from each of the Θ_i.

Ideally, we would generate Θ_i for each x_i, and then select a single θ_i from each Θ_i such that all the θ_i are consistent with one another. It is easy to test whether two IK solutions are consistent by computing the joint-angle-interpolated midpoint between the two joint configurations, and seeing if the Cartesian pose of the hand is wildly inconsistent with the average of the two Cartesian configurations. An example is in Figure B-4; the inconsistent path on top has a joint-angle-interpolated midpoint in which the hand is pointed in the entirely opposite direction.

Currently, in order to find IK solutions for our Cartesian WRT waypoints, we use an analytical IK solver to quickly solve for 6 of the 7 joint angles of the robot, given the 7th angle, and then search over the 7th angle. Although doing so is extremely fast, we currently only use this method to find a single θ_i for each x_i, rather than an entire set. This can result in an inconsistent set of θ_i.

Once we have determined that two neighboring waypoints, θ_i and θ_{i+1}, are inconsistent, we attempt to repair the inconsistency by using an incremental IK algorithm starting from one end. We use a standard constrained optimization method (Scipy.optimize.cobyla) with θ_{i+1} as the start point, and search for a consistent IK solution for θ_i. If such is found, we continue on up the path, finding a consistent θ_{i-1} starting from θ_i, and so on until we reach θ_1.

The constrained optimization function searches over possible θ, using forward kinematics to compute the resulting hand pose, x, and has a value function as follows:

$$v = d^2 + \alpha^2 + .001(\theta - \theta_0)$$

where v is the value to be minimized, d is the Cartesian distance between the desired and actual x, and α is the angle between the rotation matrices for the desired and actual x. The final term is used to penalize straying too far from the start joint angles, θ_0. The constraints are used to ensure that θ does not violate joint limits. Once the θ is found that minimizes v, we run a final, fast, incremental IK round starting from θ and without the angle penalty, to eliminate the slight bias introduced by the joint angle difference penalty.

The inconsistent path after repairing is shown in the bottom of Figure B-4. As

you can see, the new start point is not quite in the same location as the original start point; an exact consistent solution is actually infeasible for this Cartesian waypoint. However, because it is the WRT start, we allow it to deviate somewhat from the desired pose, as explained in the next section.

If no consistent set of θ_i can be found for $\tau(e)$, we declare it to be infeasible, and eliminate it from the set of action choices for the current time step. It is crucial to have a set of WRTs for which at least one action is always feasible regardless of where e is within the start uncertainty, in order for the grasp to succeed with high probability. For our experiments, even if all information-gathering and target grasps become infeasible, the reorientation grasp is always feasible, and thus the planner can attempt to reorient the object, gather information while doing so, and avoid getting stuck.

Because using the constrained optimization method for IK is much slower than using the analytical IK solver, checking whether trajectories are feasible (and finding consistent joint-angle waypoints) in this way can take quite some time. For the Brita pitcher and its seven WRTs, testing for feasiblity of all WRTs takes on average 20 seconds. Optimization IK is unnecessary 27% of the time, and the average time for those cases is only .08 seconds; for the remaining 73% of cases, feasibility testing takes 28 seconds on average.

Recently, we have switched to actually finding Θ_i for each x_i, and then selecting a consistent set of θ_i from each Θ_i, which takes only a fraction of a second for all cases, and which is a much better method in general.

B.2.3 Out of reach start locations

One unfortunate consequence of picking WRT start positions that are well out of the 95% confidence region for w is that if e is rotated significantly from the initial e for which the WRTs were designed, the WRT start position is likely to become infeasible, as in the second row of Figure B-3. However, we can use our constrained optimization IK algorithm to look for an IK solution that is as close as we can get to the desired WRT start point, and use that instead, since the start point is far enough away from the object that reaching the exact desired position and orientation is not crucial. If it is crucial to adhere exactly to the path to the next waypoint in order to avoid colliding with the object, however, having an incorrect start point can be disastrous. Thus, we can add an additional WRT waypoint closer to the object, within the collision zone. If we can find a proper IK solution for that waypoint, which is much more likely to have a proper IK solution, then the robot can get back on track, so to speak. The WRT start point thus becomes, essentially, a suggested direction to approach to avoid colliding unexpectedly with the object. As in the second row of Figure B-3, if the WRT start point is infeasible but we can find a reasonable IK solution that is not too far away from the desired configuration, and if the added WRT waypoint within the collision zone is feasible, then we allow trajectory planning to continue. If, however, even the added WRT waypoint is infeasible, as in the third row of Figure B-3, then we say that the entire trajectory is infeasible, since at that point, even if the robot knows the exact w, it may not be able to reach the goal position without colliding.

Another way to pick a reachable start point as close as possible to a desired but out-of-reach WRT start position, which we do not currently use, would be to use a fast, analytical IK algorithm to find the closest reachable point along the Cartesian path from the desired WRT start position to the next WRT waypoint. This would have the advantage of preserving as much of the desired Cartesian WRT path as possible, as well as being much faster than using an incremental IK method.

B.2.4 Cartesian interpolation

Once we have found a set of consistent joint-angle IK solutions for our WRT waypoints, we still need to be able to control the robot to move from one waypoint to the next along a Cartesian path. Because having feasible and consistent waypoints does not guarantee that the Cartesian path between them is entirely feasible, or does not pass near singularities, we still cannot just use basic, incremental Jacobian transpose control to move from one to the next. If a point along the path is barely out of reach, we would ideally like the robot to skirt along the edge of the reachable configuration space; and if the robot might need to pass near a singularity, we would like the robot to avoid getting stuck, oscillating, or flying off to someplace undesirable.

To move from the start to the end of a path segment, we compute the joint angles for closely-spaced subgoals placed along the Cartesian path, which is step 4 of Figure B-1. These subgoals, ϕ, are found by doing a joint-interpolation of the start and end of the path, and then using Jacobian transpose-based IK to adjust the joint angles to make the hand lie along the Cartesian path instead of the joint-angle path.

Jacobian transpose-based IK is an incremental IK method that uses a fast approximation to the more typical Jacobian pseudoinverse IK. Jacobian pseudoinverse IK uses the following relation:

$$\Delta\theta = J^{\dagger}\vec{e}$$

where $J^{\dagger}$ is the pseudoinverse of J, $\vec{e}$ is a magnitude-limited, 6x1 vector representing the difference between the desired and actual Cartesian hand poses, and $\Delta\theta$ is the increment to move the joint angles. Jacobian transpose IK uses the following fast approximation, which does not require expensive computation of the pseudoinverse, and which can be justified in terms of virtual forces:

$$\Delta\theta = \alpha J^{T}\vec{e}$$

where J^{T} is the transpose of the Jacobian, and alpha is some appropriate scalar. In our implementation we use a small, fixed α, but one can also select α to take appropriate variable-sized steps. More details are available in [11].

This process is shown in Figure B-5. Because Jacobian transpose-based IK can go astray when dealing with out-of-reach goals and singularities, if the resulting joint angles are farther from the Cartesian path than the interpolated joint angles, we just use the interpolated joint angles. In the worst case, even though the interpolated joint angles might deviate significantly from the Cartesian path, at least the robot will be able to get to the end point. We could instead use an incremental IK method for these failure cases that is more robust to dealing with out-of-reach or singular

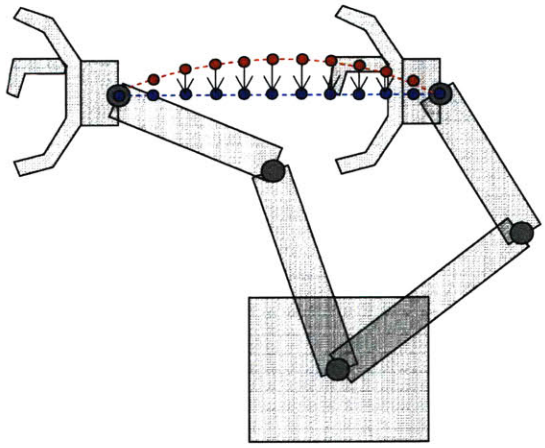

Figure B-5: Jacobian IK is used to convert joint-interpolated trajectories into Cartesian-interpolated trajectories.

cases, such as constrained optimization or selectively damped least squares [11], but as these failure cases are fairly rare, we do not currently do so.

B.2.5 Getting safely to global home positions

Different WRTs can have different global home positions. In particular, those trajectories that come from the side and those that come from above have completely separate home positions. We need a way to move between these home positions, and to get to a global home position from the robot's rest position, without colliding with the object or the robot itself regardless of where the object may be based on our start uncertainty. In order to do so, we use the motion planner OpenRAVE [14] to find a collision-free joint-angle path. To avoid colliding with the object, we place a large box obstacle in OpenRAVE's collision environment that takes up the entire volume of possible object locations, so the planner knows to avoid going through that region of space.

B.3 Trajectory control

To make our desired joint-angle path into a trajectory, we figure out where we would like the joints to be at various points in time, $\varphi(t)$. This process is step 5 of Figure B-1. We use a trajectory that is trapezoidal in velocity, as shown in Figure B-6; the robot starts from a stop, increases in velocity to a fixed maximum velocity, then decreases in velocity just before reaching the end of the entire path.

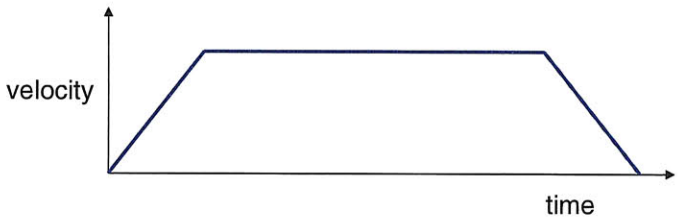

Figure B-6: A trapezoidal trajectory.

B.3.1 Generating joint-angle torques

Given a trajectory, we control the joints of the robot to move along the trajectory using joint-angle PID controllers. When the current time is equal to the next subgoal time, we switch our desired joint angles to the new subgoal. The PID controllers generate joint-angle torques at each controller time step, $\mathcal{T}(t)$, based on the error between the current actual and desired joint angles, which is step 6 of Figure B-1. At the last subgoal, we wait for the joint angles to stop moving before we declare the trajectory to be finished.

Appendix C

Geometric simulation

When computing $\Omega_\tau(w, e)$, we need to simulate what happens when the object is placed at e and the robot moves through the trajectory $\tau(e)$, and record the hand pose where the robot stops (either at first contact or at the end of the trajectory) as well as any contacts that are made. In addition, if we allow the robot to close the fingers upon making contact in order to gather additional contact information, we need to simulate that as well, so that we can estimate the value of the resulting contacts.

Because doing these simulations with the full hand and object geometries using a standard collision detection system such as Bullet, GIMPACT, PQP, or OPCODE (which is the collision detection system we use when running our physics simulations in ODE) would most likely take longer than we would like, we have our own special-purpose collision detection program.

Rather than using detailed hand meshes, we represent the hand as a set of 70 points, as shown by the teal spheres in Figure C-1. We also break down our trajectories into three separate types of motions: Cartesian translations of the hand, rotations of the hand about the center of the palm, and finger angle changes. We use the same breakdown both in the geometric simulations and when executing trajectories on the robot (simulated or real). A sample trajectory segment is shown in Figure C-2. When

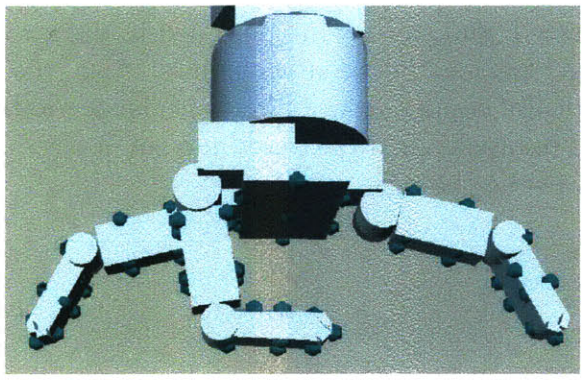

Figure C-1: Points used to represent the hand when doing geometric simulation.

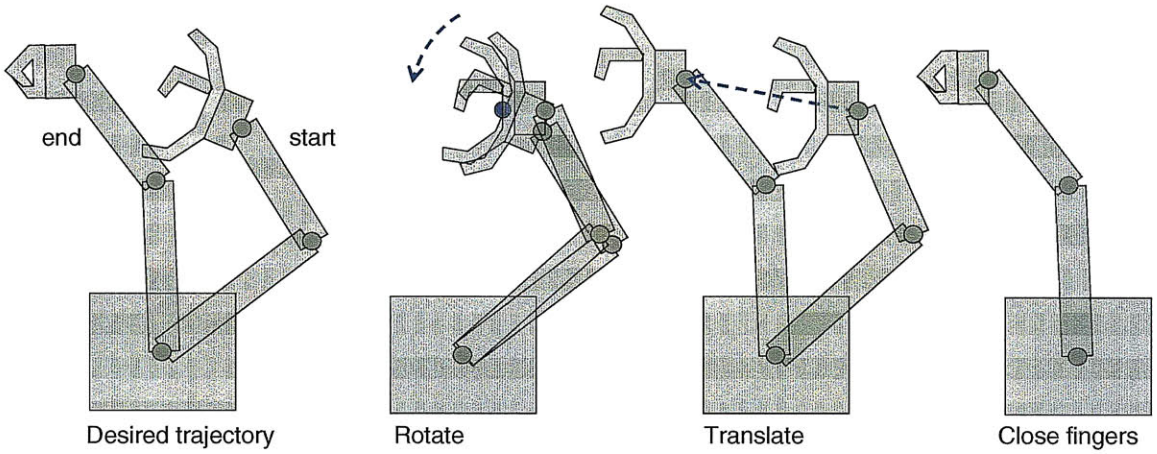

Figure C-2: Trajectory steps are separated into rotations, translations, and finger movements.

moving forward along a trajectory, for each trajectory segment, the robot rotates to the next waypoint's orientation before moving, and only closes the fingers at the goal position while holding the palm still. When backing out (after terminating a WRT), the robot does the reverse, moving before rotating.

The reason for the breakdown is that the trajectory becomes much faster to simulate geometrically. All pure translations of hand points can be seen as raytracing, which makes collision detection extremely fast. Rotations and finger angle movements are broken into short ray segments, which are still quite speedy for collision detection. The result is that simulating a grid of 24,025 different w for a given $\tau(e)$ takes only a few seconds, as opposed to many minutes.

Breaking trajectories into separate rotations and translations would appear at first glance to make using automatically-generated, collision-free, joint-angle paths difficult. However, if we break the joint-angle paths into small enough pieces (which would be required to ensure that moving in Cartesian space preserves the collision-free property of the joint-angle paths, in any event), the rotations and movements become small enough that the differences would be negligible. In our experiments, most of our WRTs are hand-generated, and so they are designed to work when broken up in this fashion even with large spacings between waypoints. Also, the few trajectories that we generated automatically happen to work fine with large rotations and translations, and so we did not need to break our trajectories into small pieces. However, one would not expect this to always be the case.

Bibliography

[1] S. Akella and M.T. Mason. Posing polygonal objects in the plane by pushing. *The International Journal of Robotics Research*, 1998.

[2] R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. *RSS*, 2007.

[3] Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *Humanoids07*, December 2007.

[4] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI*, 2005.

[5] L. Brignone and M. Howarth. A geometrically validated approach to autonomous robotic assembly. *IEEE/RSJ International Conference on Intelligent Robots and System*, 2:1626–1631, 2002.

[6] D.L Brock. Enhancing the dexterity of a robot hand using controlled slip. Master's thesis, Massachusetts Institute of Technology, 1987.

[7] R. C. Brost and A. D. Christiansen. Probabilistic analysis of manipulation tasks: A computational framework. *IJRR*, 15(1):1–23, 1996.

[8] Emma P. Brunskill. *Compact Parametric Models for Efficient Sequential Decision Making in High-dimensional, Uncertain Domains.* PhD thesis, Massachusetts Institute of Technology, 2009.

[9] B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. *ICRA*, 2007.

[10] Robert R. Burridge, Alfred A. Rizzi, and Daniel E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *I. J. Robotic Res.*, 18(6):534–555, 1999.

[11] S.R. Buss. Introduction to inverse kinematics with jacobian transpose pseudoinverse and damped least squares m ethods. Typeset manuscript, available from http://math.ucsd.edu/ sbuss/ResearchWeb, April 2004.

[12] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *IROS*, 1996.

[13] A. Censi, D. Calisi, A. De Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. *ICRA*, pages 1798–1805, 2008.

[14] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, CMU, 2008.

[15] Bruce Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37:223–271, 1988.

[16] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4:369–379, 1986.

[17] Lawrence H. Erickson, Joseph Knuth, Jason M. O'Kane, and Steven M. Lavalle. Probabilistic localization with a blind robot. *ICRA*, 2008.

[18] K. Gadeyne, T. Lefebvre, and H. Bruyninckx. Bayesian hybrid model-state estimation applied to simultaneous contact formation recognition and geometrical parameter estimation. *IJRR*, 24:615, 2005.

[19] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artif. Intell.*, 147(1-2):163–223, 2003.

[20] J. P. Gonzalez and A. Stentz. Planning with uncertainty in position an optimal and efficient planner. *IROS*, pages 2435–2442, 2005.

[21] R.A. Grupen and Jr. J.A. Coelho. Acquiring state from control dynamics to learn grasping policies for robot hands. *Advanced Robotics*, 16(5):427–443, 2002.

[22] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez. Robust belief-based execution of manipulation programs. In *WAFR*, 2008.

[23] Kaijen Hsiao, Paul Nangeroni, Manfred Huber, Ashutosh Saxena, and Andrew Ng. Reactive grasping using optical proximity sensors. In *ICRA*, 2009.

[24] D. Kragic and H.I. Christensen. Survey on visual servoing for manipulation. *IEEE Transaction on Robotics & Automation*, 15(2):238–250, 1999.

[25] A. Krause and C. Guestrin. Near-optimal observation selection using submodular functions. In *Proceedings of the National Conference on Artificial Intelligence*, 2007.

[26] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *RSS*, 2008.

[27] J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, Mass, 1991.

[28] Jean-Claude Latombe, Anthony Lazanas, and Shashank Shekhar. Robot motion planning with uncertainty in control and sensing. *Artif. Intell.*, 52(1):1–47, 1991.

[29] S. M. LaValle and S. A. Hutchinson. An objective-based stochastic framework for manipulation planning. In *IROS*, pages 1772–1779, September 1994.

[30] SM LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26(3):430–465, 2000.

[31] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006, to appear.

[32] T. Lefebvre, H. Bruyninckx, and J. DeSchutter. Polyhedral contact formation identification for autonomous compliant motion: Exact nonlinear bayesian filtering. *IEEE Transactions on Robotics*, 21(1):124–129, 2005.

[33] Tomás Lozano-Pérez, Matthew Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[34] TB Martin, RO Ambrose, MA Diftler, R. Platt, and MJ Butzer. Tactile gloves for autonomous grasping with the nasa/darpa robonaut. In *IROS*, 2004.

[35] Brian Mayton, Eric Garcia, Louis LeGrand, and Joshua R. Smith. Electric field pretouch: Towards mobile manipulation. In *RSS Workshop on Mobile Manipulation in Human Environments*, 2009.

[36] Brenan J. McCarragher and Haruhiko Asada. A discrete event approach to the control of robotic assembly tasks. In *ICRA*, pages 331–336, 1993.

[37] N. A. Melchior and R. Simmons. Particle rrt for path planning with uncertainty. *ICRA*, 2007.

[38] Andrew Miller and Peter K. Allen. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11(4):110–122, Dec 2004.

[39] Wyatt S. Newman, Michael Branicky, Yoh-Han Pao, Craig Birkhimer, Siddharth Chhatpar, Jing Wei, and Yonghong Zhao. Intelligent strategies for compliant robotic assembly. In *Proc. 11th Yale Workshop on Adaptive and Learning Systems*, pages 139–146, 2001.

[40] I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: An office-navigating robot. *AI magazine*, 16(2):53–60, 1995.

[41] Sylvie C. W. Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Pomdps for robotic tasks with mixed observability. In *RSS*, 2005.

[42] A. Petrovskaya and A. Y. Ng. Probabilistic mobile manipulation in dynamic environments, with application to opening doors. *IJCAI*, 2007.

[43] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, 2003.

[44] R. Platt. Learning Grasp Strategies Composed of Contact Relative Motions. In *IEEE-RAS International Conference on Humanoid Robots*, 2007.

[45] R. Platt Jr, AH Fagg, and RA Grupen. Nullspace composition of control laws for grasping. In *IROS*, volume 2, 2002.

[46] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. *ISRR*, 2007.

[47] N. Roy and S. Thrun. Coastal navigation with mobile robot. In *NIPS99*, 2000.

[48] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *IJCAI*, pages 1080–1087, 1995.

[49] M. Skubic and R. A. Volz. Acquiring robust, force-based assembly skills from human demonstration. *IEEE Transactions on Robotics and Automation*, 16(6):772–781, 2000.

[50] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[51] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press.

[52] M.T.J. Spaan and N. Vlassis. Perseus: randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[53] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[54] Georgios Theocharous and Leslie Pack Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems 16 (NIPS03)*, 2004.

[55] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.