MIT-CSAIL-TR-2012-019 July 3, 2012

Integrated robot task and motion planning in belief space

Leslie Pack Kaelbling and Tomas Lozano-Perez

Integrated robot task and motion planning in belief space

Leslie Pack Kaelbling and Tomás Lozano-Pérez
CSAIL, MIT, Cambridge, MA 02139
{lpk, tlp}@csail.mit.edu

Abstract

In this paper, we describe an integrated strategy for planning, perception, state-estimation and action in complex mobile manipulation domains. The strategy is based on planning in the *belief space* of probability distribution over states. Our planning approach is based on hierarchical goal regression (pre-image back-chaining). We develop a vocabulary of fluents that describe sets of belief states, which are goals and subgoals in the planning process. We show that a relatively small set of symbolic operators lead to task-oriented perception in support of the manipulation goals.

An implementation of this method is demonstrated in simulation and on a real PR2 robot, showing robust, flexible solution of mobile manipulation problems with multiple objects and substantial uncertainty.

1 Introduction

As robots become more capable of sophisticated sensing, navigation, and manipulation, we would like them to carry out increasingly complex tasks autonomously. A robot that helps in a household must select actions over the scale of hours or days, considering abstract features such as the desires of the occupants of the house, as well as detailed geometric models that support locating and manipulating objects. The complexity of such tasks derives from very long time horizons, large numbers of objects to be considered and manipulated, and fundamental uncertainty about properties and locations of those objects. Specifying control policies directly, in the form of tables or state machines, becomes intractable as the size and variability of the domain increases. However, good control decisions can be made with a compact specification by using a model of the world dynamics to do on-line planning and execution.

The uncertainty in problems of this kind is pervasive and fundamental: it is, in general, impossible to sense sufficiently to remove all of the important uncertainty. The robot will not know the contents of all of the containers in a house, or where someone left their car keys, or the owner's preference for dinner. In order to behave effectively in the face of such uncertainty, the robot must explicitly take actions to gain information: look in a cupboard, remove an occluding object, or ask someone a question.

We have developed an approach to integrated task and motion planning that integrates geometric and symbolic representations in an aggressively hierarchical planning architecture, called HPN (for *hierarchical planning in the now*), which we summarize in section 3 and discuss in detail in [Kaelbling and Lozano-Pérez, 2011, 2012]. The hierarchical decomposition allows efficient solution of problems with very long horizons; the symbolic representations support abstraction in complex domains with large numbers of objects and are integrated effectively with the detailed

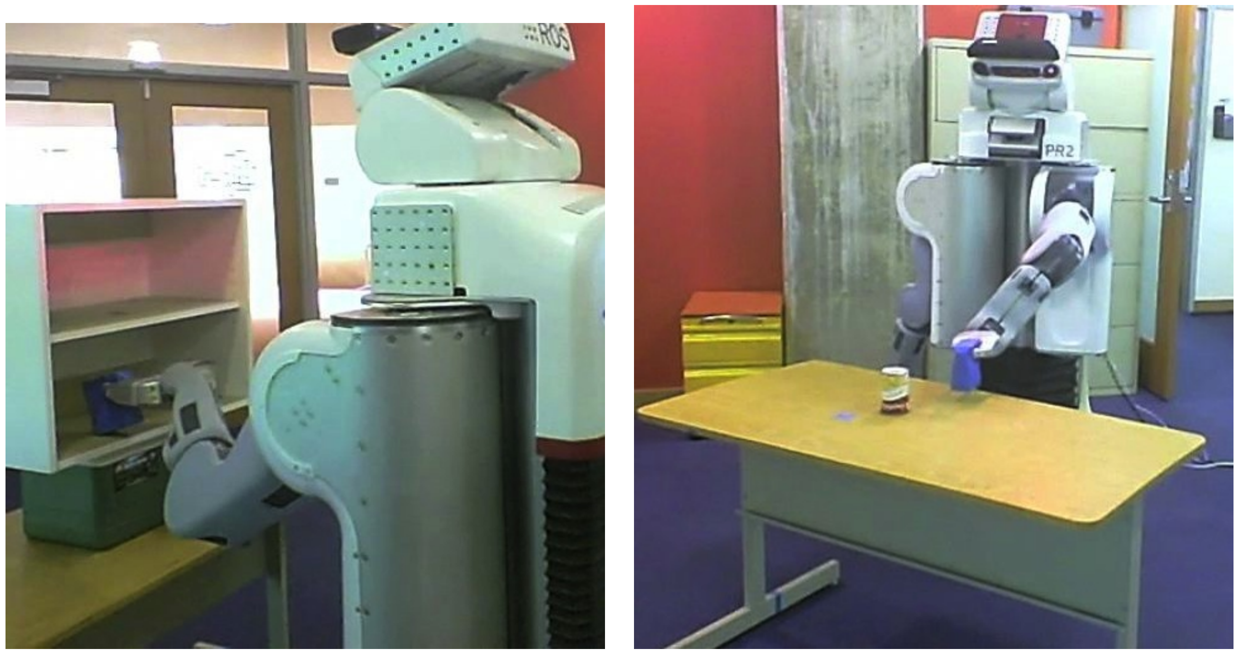


Figure 1: PR2 robot manipulating objects

geometric models that support motion planning. In this paper, we extend the HPN approach to handle two types of uncertainty: *future-state* uncertainty about what the outcome of an action will be, and *current-state* uncertainty about what the current state actually is. Future-state uncertainty is handled by planning in approximate deterministic models, performing careful execution monitoring, and replanning when necessary. Current-state uncertainty is handled by planning in *belief space*: the space of probability distributions over possible underlying world states. Explicitly modeling the robot's uncertainty during planning enables the selection of actions based both on their ability to gather information as well as their ability to change the state of the world. This belief-space approach enables seamless integration of action and perception.

This paper describes a tightly integrated approach, weaving together perception, estimation, geometric reasoning, symbolic task planning, and control to generate behavior in a real robot that robustly achieves tasks in complex, uncertain domains. We have formulated this method in the context of a mobile manipulator doing household tasks, such as the one shown in figure 1, but it can be applied much more broadly. Problems of surveillance or locating objects of interest are naturally formulated in this framework, as are more complex operational tasks.

1.1 Handling uncertainty

The decision-theoretic optimal approach to planning in domains with probabilistic dynamics is to make a *conditional plan*, in the form of a tree or policy, supplying an action to take in response to any possible outcome of a preceding action. Figure 2 illustrates one strategy, which consists of building a tree starting at the current world state s, branching on the robot's choice of actions a and then, for each action, branching on the probability of each resulting state s'. To select actions, one grows the tree out to some depth k, then evaluates it from the bottom up using the "expectimax" rule. A static evaluation function assigns a value ρ_0 to each leaf node. Then, the

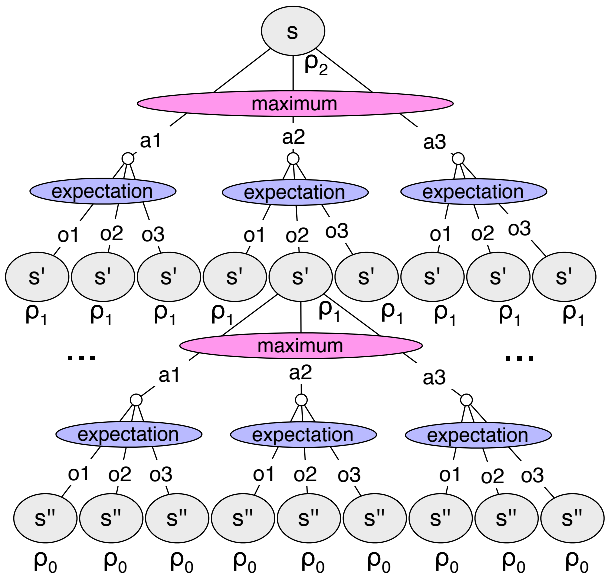


Figure 2: Probabilistic policy tree

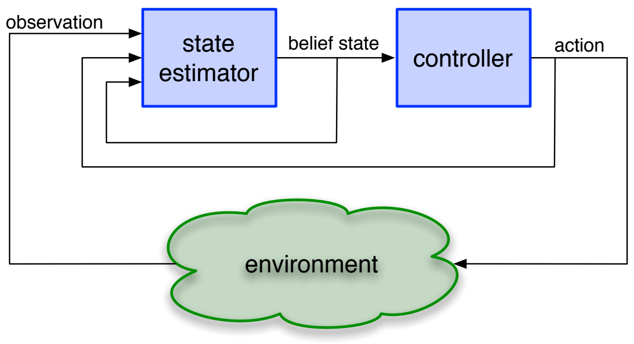


Figure 3: POMDP estimation and control architecture

value of a probabilistic node is computed as the expected value of its children, and the value of an action-choice node is compute as the maximum of the values of its children. Now, the policy consists of selecting the maximizing action at any action-choice node that is reached.

The process of constructing and evaluating a tree of this kind can be prohibitively expensive; but there have been recent advances in effective sample-based approaches [Gelly and Silver, 2008]. For efficiency and robustness, our approach to action selection is to construct a deterministic approximation of the dynamics, use the approximate dynamics to build a sequential (non-branching) plan, execute the plan while perceptually monitoring the world for deviations from the expected outcomes of the actions, and replan when deviations occur. This method has worked well in control applications [Mayne and Michalska, 1990] as well as symbolic planning domains [Yoon et al., 2007].

Replanning approaches that work in the state space handle uncertainty in the future outcomes of actions, but not uncertainty about the current world state. Current-state uncertainty is unavoidable in most real-world applications. The optimal solution to such problems is found in the formulation of partially observable Markov decision processes (POMDPs) [Smallwood and Sondik, 1973] and involves planning in the *belief space* rather than the underlying state space of the domain. The belief space is the space of probability distributions over underlying world states. A controller in such a problem is divided into two components, as shown in figure 3. The *state estimator* is a process (such as a Bayesian filter or a Kalman filter), which maintains a current distribution over underlying world states, conditioned on the history of actions and observations the system has made. The *controller* is policy, a mapping from belief states to actions: it can be computed off-line and stored in a representation that allows efficient execution, or it can itself be a program that does significant computation on-line to select an appropriate action for the current belief state.

In the traditional POMDP literature, the goal is to find an optimal or near-optimal policy using off-line computation; the policy can then be executed on-line with little further computation, simply determining which of a finite number of categories the belief state belongs to, and executing the associated action. Recent point-based solvers [Kurniawati et al., 2008] can find near-optimal policies for domains with thousands of states. However, the problem of computing a near-optimal policy for uncertain manipulation domains with many movable objects is still prohibitively expensive.

Our strategy will be to construct a policy "in the now": that is, to apply the HPN approach to interleaved planning and execution, but in the space of beliefs, using a determinized version of the belief-space dynamics. Belief space is continuous (the space of probability distributions) and so is not amenable to standard discrete planning approaches. We address it with HPN in the same way that we address planning for geometric problems: by dynamically constructing discretizations that are appropriate for the problem at hand.

We will use symbolic descriptions to characterize aspects of the robot's belief state to specifying goals and subgoals during planning. For example, the condition "With probability greater than 0.95, the cup is in the cupboard," can be written as $BIn(cup, cupboard, 0.05)$, and might serve as a goal for planning. Our description of the effects of the robot's actions, encoded as operator descriptions, will not be in terms of their effect on the state of the external world, which is not observable, but in terms of their effect on the logical assertions that characterize the robot's belief. In general, it will be very difficult to characterize the exact pre-image of an operation in belief space; we will strive to provide an approximation that supports the construction of reasonable plans and rely on execution monitoring and replanning to handle errors due to approximation. We will describe and illustrate this approach in detail in the rest of the paper.

1.2 Related work

Advances in perception, navigation and motion planning have enabled sophisticated manipulation systems that interleave perception, planning and acting in realistic domains (e.g., [Srinivasa et al., 2009, Rusu et al., 2009, Pangercic et al., 2010]). Although these systems use various forms of planning, there is no systematic planning framework that can cope with manipulation tasks that involve abstract goals (such as "cook dinner"), that can plan long sequences of actions in the presence of substantial uncertainty, both in the current state and in the effect of actions, and that can plan for acquiring information.

Dogar and Srinivasa [2012] comes closest among existing systems to our goals. They tackle manipulation of multiple objects using multiple types of primitives, including grasping and pushing. The inclusion of pushing enables very efficient solutions to some problems of manipulation in cluttered environments. They also include explicit modeling of uncertainty in object poses and the effect of the actions on this uncertainty. Their implementation does not currently encompass actions aimed at gathering information, but it appears that their approach could readily be extended to such actions. A fundamental difference from our approach is that they plan at the level of object poses and do not integrate task-level reasoning over higher-level goals. Also, since their approach does not reason hierarchically, it will have difficulty extending to tasks requiring a very long sequence of actions to accomplish.

There have been several recent approaches to integrated task and motion planning [Cambon et al., 2009, Plaku and Hager, 2010, Marthi et al., 2010, Dornhege et al., 2009] with the potential of scaling to mobile manipulation problems; our companion paper [Kaelbling and Lozano-Pérez, 2012] reviews this related work in detail. However, none of these approaches addresses uncertainty directly.

A number of recent papers [Alterovitz et al., 2007, Levihn et al., 2012] tackle motion planning problems with substantial future-state uncertainty. Most relevant here is the work of Levihn et al. [2012], which tackles the problem of navigating among movable obstacles in the presence of uncertainty in the effect of robot actions on the obstacles. However, this work assumes that there is no uncertainty in the current state. Although the pervasive impact of current-state uncertainty in robotics has motivated an enormous body of work on perception and control, the work on planning in the presence of current-state uncertainty is more limited.

There have been attempts to formulate and solve problems of planning robot motions under uncertainty in non-probabilistic frameworks dating back to the "preimage backchaining" paper [Lozano-Pérez et al., 1984]. Work following on this line seeks to construct strategies, based on information sets, that are guaranteed to reach a goal and to know that they have reached it, with guaranteed termination conditions [Brafman et al., 1997, Erdmann, 1994, Donald, 1988]. Excellent summaries of subsequent work on planning with uncertainty are available [Latombe, 1991, LaValle, 2006].

Within the probabilistic setting, there is a history of formulating mobile-robot navigation problems as POMDPs, beginning with simple heuristic solution methods [Cassandra et al., 1996, Simmons and Koenig, 1995] and progressing to more sophisticated approaches; a summary of this work can be found in [Thrun et al., 2005]. Recent point-based POMDP solvers [Kurniawati et al., 2008] can be leveraged to solve a variety of robotics problems [Hsiao et al., 2007, Ong et al., 2010, Kurniawati et al., 2011], but mobile manipulation problems involving many movable objects are beyond the scope of these methods. Under some restricted assumptions, suitable for mobile-robot navigation and control, much more efficient solutions are possible [Prentice and Roy, 2007, van den Berg et al., 2011]. One approach to addressing large POMDPs is to find policies, but to do so in large spaces by exploiting structural regularities described using Bayesian-network, decision-diagram, or even first-order representations of the underlying domain [Sanner and Kersting, 2010, Wang and Khardon, 2010].

Our approach in this paper can be seen as finding approximate solutions to very large POMDP problems by planning in belief space using simplified models and re-planning. Recent research [Platt et al., 2010, Erez and Smart, 2010, du Toit and Burdick, 2010, Hauser, 2010] has established the value of control in belief space using simplified models and replanning. Our approach to belief space planning builds directly on this work.

Within the AI planning community, there has been a great deal of work on planning in a belief space corresponding to sets of underlying world states [Bryce et al., 2006, Bertoli et al., 2001]; here we concentrate on approaches that use explicit logical formulation of knowledge conditions. There is also a long history of using logical representations to formalize knowledge preconditions for planning, starting informally with the work of McCarthy and Hayes [1969], then formalized by Moore [1985] and Morgenstern [1987]. This approach has been implemented in a planning system by Petrick and Bacchus [2004] and is used in conjunction with a logic-programming based reasoning and behavior-specification mechanism in Flux [Thielscher, 2005].

Regression-based planning is used in several ways that are related to ours. Fritz and McIlraith [2009b] find plans that are robust to changes during execution and in [Fritz and McIlraith, 2009a] perform regression-based planning and have a novel method for execution monitoring for changes that happen *during* planning. Tuan et al. [2006] provide a regression method that generates plans in partially observable domains that include reasoning about knowledge and explicitly include sensing actions. Howver these systems do not use a probabilistic model of uncertainty or extend to continuous domains such as robot motion planning.

1.3 Paper outline

Section 2 describes the robotic manipulation problem that has inspired this work and that serves as both a simulated and physical test domain. Section 3 provides a summary of the HPN approach in domains without uncertainty, including the combination of logical and geometric reasoning and the representation of the domain dynamics in terms of operator descriptions. We attempt to provide basic definitions as we go to make this paper reasonably clear on its own. Section 4 describes our approach to handling uncertainty in future outcomes and in the current state; we define BHPN, which is the HPN approach in belief space. Section 5 explains, in detail, how a logical formulation can be applied to belief space, first in two simple example cases and then in the context of robot manipulation. Section 6 provides operator descriptions for manipulation and sensing operations in the robot domain. Section 7 demonstrates BHPN running in several example problems illuminating the ability to plan to gain information and the interweaving of estimation, planning, and execution to gain robustness in uncertain domains.

2 Example problem domain

The methods described in this paper are designed to apply broadly to robotics applications involving uncertainty in complicated domains. To make the discussion concrete, we will use a problem domain of picking and placing objects using a Willow Garage PR2 robot. Some simple examples in this domain have been executed on the real robot and are illustrated in section 7 and in the companion videos that may be found at `http://people.csail.mit.edu/tlp/IJRRBel.html`.

2.1 Perception and control

The robot can move its 7-DOF left arm, 2-DOF head, 1-DOF gripper and 3-DOF base. There is significant error in the base odometry, much less error in the arm positioning relative to the base, and negligible error in the head pose. For simplicity in grasp selection and efficient trajectory planning, we limit the robot to grasps that keep its hand parallel to the floor; this is merely expedient and is in no way a necessary feature of the approach.

We assume that all of the objects are unions of convex polyhedra that are extrusions along the z axis (that is, they have a constant horizontal cross-section) and that their shapes are known in advance. The world state is not dynamic, in the sense that objects are always at rest unless they are in the robot's gripper and the robot is moving. We assume that all objects that are not in the hand are resting stably on a horizontal surface; this means that their poses can be represented in four dimensions: x, y, z, and θ (which is rotation about the z axis). We also assume that objects do not touch each other, due to limitations on the segmentation abilities of the current perception system.

The robot can observe the environment with the stereo sensors on its head, which can be panned and tilted. The sensors yield three-dimensional point clouds; a perception service attempts to detect instances of the known shape models in a given point cloud. This is done by locating horizontal planes in the point cloud, finding clusters of points resting on the surface, and then doing stochastic gradient descent over the space of poses of the models to find the pose that best matches the group. We use the current estimated poses and uncertainties of the objects to limit which models are matched to which point clusters.

There is error in the detected poses of the objects but, at present, we assume that there is no identity uncertainty, that is, the perceived identities of objects are correct and unique. The robot also has a scanning laser on its torso that produces a point cloud with much larger coverage but lower spatial resolution. This point cloud is used for detecting support surfaces (tables) in the workspace.

In our examples, the robot picks and places objects, relying primarily on visual sensing. To further increase robustness in grasping objects, we also employ a simple reactive grasping procedure that uses information from touch sensors on the fingers to recover from small errors in the object's location estimates.

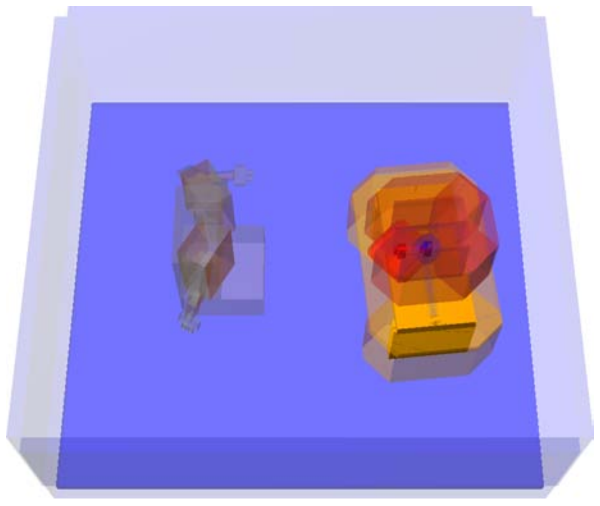
2.2 Belief state representation and update

The robot updates its representation of the belief state after every action (physical or perceptual). The belief state for mobile manipulation under uncertainty needs to contain enough information to support queries both about the nominal (mean or mode) state of the world and about the system's confidence in those estimates. The confidence estimates must be accurate enough to support decision-theoretic trade-offs between, for example, attempting to pick up an object or taking another look to localize it more accurately. It also needs to be a sufficient statistic for the history of observations so that recursive updates may be made effectively. We do not believe there is a good uniform representational strategy for all aspects of information about the domain, so we have separate representations for poses of known objects and for observed space.

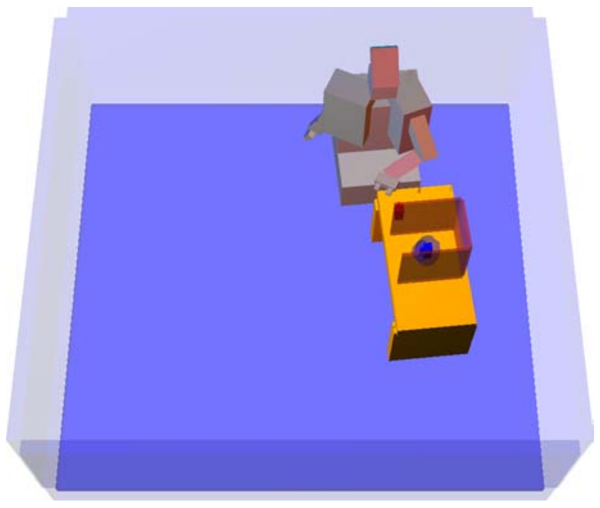
2.3 Object poses

When execution is initiated, the robot knows of the existence of some (but possibly not all) objects in the domain, and knows what their shapes are; it may have a very highly uncertain estimate of their poses. Figure 4(a) shows an initial belief state, in terms of 0.05-shadows of the objects. Shadows are defined in detail in section 6.2.1: an ϵ-shadow of an object can be understood as the volume of space that, with probability greater than $1 - \epsilon$ contains all parts of the object. In this example there is a table with two squared-off cups on it (one red and one blue) and a u-shaped 'cupboard'. There is substantial uncertainty about their poses; their shadows are drawn in the same basic color and indicate the geometric extent of the uncertainty.

Because our demonstrations currently involve relatively few objects, we simply represent the distribution over their poses, together with the base pose of the robot, using a full joint Gaussian



(a) Initial belief state for simple problem of moving the red cup.



(b) Final belief state for moving the red cup, satisfying the goal that, with high probability, the cup is contained in a region on the left side of the table.

Figure 4: Example starting and ending belief states.

distribution. Since the shapes of objects are completely known and they are always resting stably on a known face, they have four degrees of pose freedom: x, y, z, and θ.

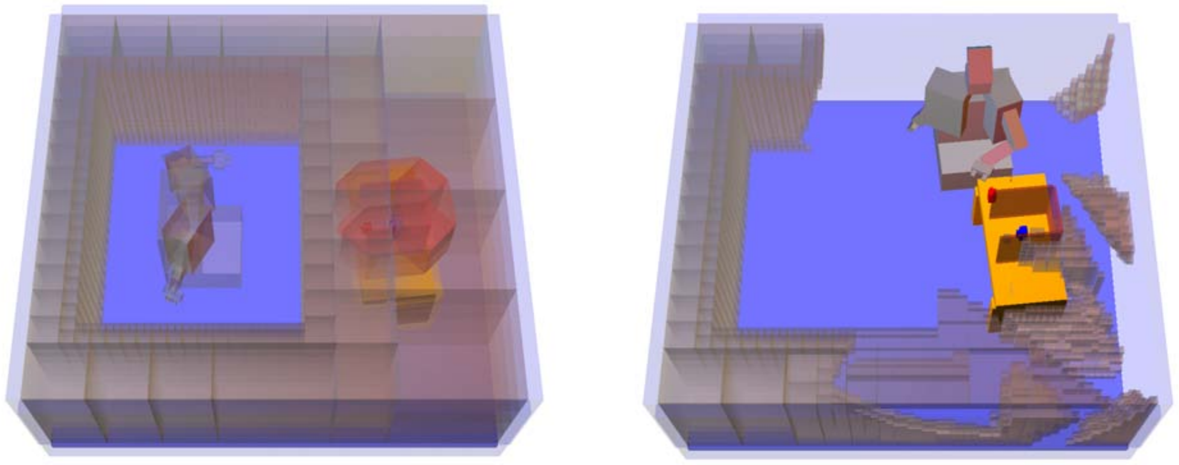
When the robot moves, an odometry error model is used to do an unscented Kalman filter (UKF) [Julier and Uhlmann, 2004] update of the belief, with a control input computed as the difference between the uncorrected raw odometry of the robot at the current time and at the time of the previous update. When an object is detected in the perceptual input, it is rejected as an outlier if its detected pose relative to the robot is highly unlikely in the current model; otherwise, the detection, reported in robot-relative coordinates, is also used in a UKF update, which affects the robot and object poses, as well as other poses through entries in the covariance matrix. Observation noise is assumed to be Gaussian, and there is no handling of false negative observations.

Care must be taken to estimate poses correctly when there are angular dimensions. We handle this by using a *wrapped Gaussian* representation for the angular dimension, essentially constructing a real space tangent to the unit circle at the mean of the angular distribution. This approach is convenient because the product can be taken of multiple tangent spaces together with regular real spaces for the other dimensions, and a single multivariate Gaussian used to represent the entire joint distribution over the product space [Fletcher et al., 2003].

An additional concern during estimation is the incorporation of physical constraints: objects may not interpenetrate one another and must be supported by other objects (not be floating in the air). The estimator we use in this paper does not address these constraints, except for enforcing the z positions of objects so that they rest on top of a supporting surface, in an *ad hoc* way. In parallel work, we have developed an approach to solving this problem [Wong et al., 2012] which has not yet been integrated with the system reported here.

The belief state is augmented with a point estimate of the arm and head configuration of the robot, the object that is currently being grasped by the robot, if any, and the grasp by which it is being held.

Figure 4(b) shows the terminal belief state from an execution trace that started with the belief



(a) Initial belief state for simple problem of moving the red cup showing the unobserved spaced filled with the gray cells of the oct-tree.

(b) Final belief state for moving the red cup, satisfying the goal that, with high probability, the cup is contained in a region on the left side of the table. The observed space oct-tree reflects the observations during the operation.

Figure 5: Example starting and ending belief states including the observed space oct-tree.

state in figure 4(a) and had the goal of believing that the red cup was placed within a region of space on the left side of the table. Although it is somewhat occluded by the cupboard in this view, we can see that the red cup has been moved into the desired region and has very low positional uncertainty (it has a very small shadow). In contrast, we can see that the blue cup, which was irrelevant to this problem, has not had its position well localized. By planning in belief space, the system controls the sensing process, executing only enough sensing actions to support the achievement of the overall goal.

2.4 Observed space

Another important query that must be supported by the belief state is whether a region of space is known to be clear of obstacles and therefore safe to traverse. To answer such queries, we represent the parts of the space that the robot has recently observed with its depth sensors.

Keeping an accurate probabilistic model of known-clear space is quite difficult: the typical approach in two-dimensional problems is an occupancy grid [Elfes, 1989] (recently extended to three-dimensional problems [Wurm et al., 2010]). It requires a detailed decomposition of the space into grid cells and although there are some attempts to handle odometry error in the robot [Souza et al., 2008], this remains challenging. A more principled strategy would be to maintain the history of robot poses in the UKF, rather than just the current one, and to combine the depth maps sensed at each of those poses into a global map of observed space.

We take a much simpler approach, operating under two assumptions: first, that the observed-space maps we construct will only be used in the short term; and, second, that the mechanisms for detecting objects and tracking their poses will provide a high-accuracy estimate of the poses of material objects. Looking is not too expensive, and objects may be dynamic, so we expect, for instance, when the robot re-enters a room, that it will need to reconstruct the observed-space map. Thus, handling long-distance relative odometry errors is not crucial. For this reason, we simply attach each depth scan to the most likely pose estimate for the robot in the UKF at the time the scan was taken (this is much more accurate than the raw odometry). We integrate the observed-space information from the sequence of scans into an oct-tree representation of the space that has been observed by the robot. This representation of observed space need not be as high-resolution as an occupancy grid, which must also represent the boundaries between free and occupied regions of the environment; in our approach, those boundaries are represented by the continuous object-pose distributions in the UKF.

In the following sections, we will denote space that has been observed as $\mathcal{S}_{obs}$. Figure 5 shows the observed-space oct-tree at two points during an execution; the space that is filled with dark gray cells has not yet been observed by the robot. At initialization time (figure 5(a)), the robot knows the contents of the region of space right around it. As it moves and scans (in this case, using both the scanning laser on the torso as well as the narrow-field stereo cameras on the head), it clears out more of the space, until in the final frame (figure 5(b)), it has observed much of the observable space in the room. One very important role that the observed-space map plays is to constrain the robot motion planner when it is determining a trajectory for final execution of a motion primitive; any part of the space that has not yet been observed is marked as an obstacle and cannot be traversed.

3 HPN in observable domains

This section has been drawn in its entirety from [Kaelbling and Lozano-Pérez, 2012] in order to make this paper more self contained.

HPN is a hierarchical approach to solving long-horizon problems, which performs a temporal decomposition by planning operations at multiple levels of abstraction; this ensures that problems to be addressed by the planner always have a reasonably short horizon, making planning feasible.

In order to plan with abstract operators, we must be able to characterize their preconditions and effects at various levels of abstraction. Even at abstract levels, geometric properties of the domain may be critical for planning; but if we plan using abstracted models of the operations, we will not be able to determine a detailed geometric configuration that results from performing an operation. To support our hierarchical planning strategy we, instead, plan backwards from the goal using the process known as *goal regression* [Ghallab et al., 2004] in task planning and *pre-image backchaining* in motion planning [Lozano-Pérez et al., 1984]. Starting from the set of states that satisfies the goal condition, we work backward, constructing descriptions of pre-images of the goal under various abstract operations. The pre-image is the set of states such that, if the operation were executed, a goal state would result. The key observation is that, whereas the description of the detailed world state is an enormously complicated structure, the descriptions of the goal set, and of pre-images of the goal set, are often describable as simple conjunctions of a few logical requirements.

In a continuous space, pre-images might be characterized geometrically: if the goal is a circle of locations in x, y space, then the operation of moving one meter in x will have a pre-image that is also a circle of locations, but with the x coordinate displaced by a meter. In a logical, or combined logical and geometric space, the definition of pre-image is the same, but computing pre-images will require a combination of logical and geometric reasoning. We support abstract geometric reasoning by constructing and referring to salient geometric objects in the logical representation used by the planner. So, for example, we can say that a region of space must be clear of obstacles before an object can be placed in its goal location, without specifying a particular geometric arrangement of the obstacles in the domain. This approach allows a tight interplay between geometric and non-geometric aspects of the domain.

The complexity of planning depends both on the horizon and the branching factor. We use hierarchy to reduce the horizon, but the branching factor, in general, remains infinite: there are innumerable places to put an object, innumerable paths for the robot to follow from one configuration to another, and so on. We use *generators*, functions that make use both of constraints from the goal and heuristic information from the starting state to make choices from these infinite domains that are likely to be successful; our approach can be extended to support sample-based backtracking over these choices if they fail. Because the value-generation process can depend on the goal and the initial state, the values are much more likely to be successful than ones chosen through an arbitrary sampling or discretization process.

Even when planning with a deterministic model of the effects of actions, we must acknowledge that there may be errors in execution. Such errors may render the successful execution of a very long sequence of actions highly unlikely. For this reason, we interleave planning with execution, so that all planning problems have short horizons and start from a current, known initial state. If an action has an unexpected effect, a new plan can be constructed at not too great a cost, and execution resumed. We refer to this overall approach as HPN for "hierarchical planning in the now."

In the rest of this section, we provide a more detailed overview of the HPN approach. We use a simple, one-dimensional environment to illustrate the key points.

3.1 Representing objects, relations and actions

Figure 6 shows three configurations of an environment in which there are two blocks, a and b, on a table. They are each 0.5 units wide, and their position is described by the (continuous) position of their left edges. The black line in the figure represents the table and the boxes beneath the line represent regions of one-dimensional space that are relevant to the planning process. The configuration space, then, consists of points in $\mathcal{R}^2$, with one dimension representing the left edge of a and the other representing the left edge of b. However, some (l_a, l_b) pairs represent configurations in which the blocks are in collision, so the free configuration space is $\mathcal{R}^2$ with the illegal locus of points $\{(l_a, l_b) \mid l_a - l_b < 0.5\}$, which is a stripe along the diagonal, removed.

At the lowest level of abstraction, the domain is represented in complete detail, including shapes and locations of all objects and the full configuration of the robot. This representation is used to support detailed motion planning.

At higher levels of abstraction, we use logical representations to characterize both geometric and non-geometric aspects of the world state. A *fluent* is a condition that can change over time, typically expressed as a logical predicate applied to a set of arguments, which may be constants or variables. To describe our example domain, we use instances of the following fluents:

- $In(o, r)$ indicates that object o is completely inside the region r;
- $ObjLoc(o, l)$ indicates that the left edge of object o is at location l; and,
- $ClearX(r, x)$ indicates that no objects overlap region r, except possibly for objects named in the set x.

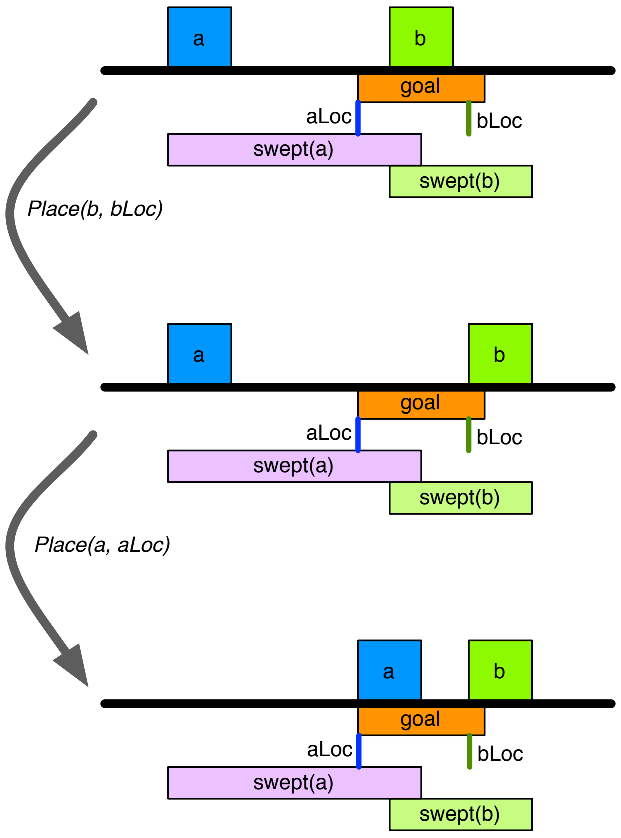


Figure 6: Simple one-dimensional environment: starting configuration, intermediate configuration after moving b out of the way, final configuration with a in the goal region.

```
In(o, r):
    result: $In(o, r)$
    choose: $l \in$ generateLocsInRegion$(o, r)$
    pre: $ObjLoc(o, l)$

Place(o, l_target):
    result: $ObjLoc(o, l_{target})$
    choose: $l_{start} \in \{currentLoc(o),$ generateLocsInRegion$(o, warehouse)\}$
    pre: $ObjLoc(o, l_{start})$
         $ClearX(sweptVol(o, l_{start}, l_{target}), (o))$

Clear(r, x):
    result: $ClearX(r, x)$
    pre: $In(o, U \setminus r)$ for $o \in (allObjects \setminus x)$
```

Figure 7: Simplified operators used in the example.

Note that by “logical” representations, we mean representations that are made up of explicit references to entities in the domain (including objects, regions, paths, and poses), relations among these entities (including object *in* region, or pose *near* object) that have truth values in a given state, and Boolean combinations of these relations. We emphatically do not mean that these representations are free of numbers, as the examples above attest. The key benefit of logical representations is that they give us compact representations of large, possibly infinite, sets of world states and of the dynamic effects of actions on such large or infinite domains. Our key requirement on the representations will be that, given a logical description, we must be able to determine whether a geometric world state satisfies that description. For example, we must be able to test whether $In(A, goal)$ is true of a world state.

To complete a domain representation, we must also describe the available actions. We will use *operators* which are loosely modeled on the operators of classical task planning. An operator describes the *preconditions* of an action and its *effects*. In our simple environment, we have three types of operators, each of which has as its effect one of the types of assertions mentioned above; figure 7 shows them written in the syntax we use throughout the paper. The first one actually causes a physical effect in the environment; the other two are essentially inference rules that reduce one logical condition to another.

- $\textsc{Place}(o, l_{target})$ places object o at a target location l_{target}; the effect is $ObjLoc(o, l_{target})$. The preconditions are that the object be in some location, l_{start}, before the operation takes place and that the region “swept” by the object moving from l_{start} to l_{target} be clear except for o, that is, $ClearX(sweptVol(o, l_{start}, l_{target}), \{o\})$.

- $\textsc{In}(o, r)$: the effect is $In(o, r)$ if $ObjLoc(o, l)$ is true for some location, such that the volume of object o when located at l is contained in region r.

- $\textsc{Clear}(r, x)$: the effect is $ClearX(r, x)$ if all objects in the universe except those in the set x do not overlap region r; that is, that for all objects o not in x, $In(o, \overline{r})$, where $\overline{r}$ is the spatial complement of r (that is, the spatial domain minus r).

3.2 Regression and generators

In HPN, the goal of the system is to reach some element in a set of states of the world, described using a logical expression. So, for example, a goal might be for object a to be in the region *goal*, which would be written $In(a, goal)$. The planning algorithm proceeds by applying a formal description of the operations in the domain to find one that might achieve the goal condition, then adopting the preconditions of that operation as a new sub-goal. This process of goal-regression proceeds until all of the conditions in the subgoal are true in the current world state.

To illustrate regression and generation in HPN, we work through a very simple example in detail. The goal of our simple planning problem is for block a to be in the goal region, which is indicated in orange. There is no explicit robot in this environment: to keep the space low-dimensional, we will simply consider motions of the objects along the one-dimensional table.

Figure 8 shows a path in the regression search tree through sets of configurations of the blocks. Recall that the configuration space in this domain is the positions of the left edges of the two blocks, which is $\mathcal{R}^2$, minus the configurations along the diagonal in which the objects would be in collision with one another.

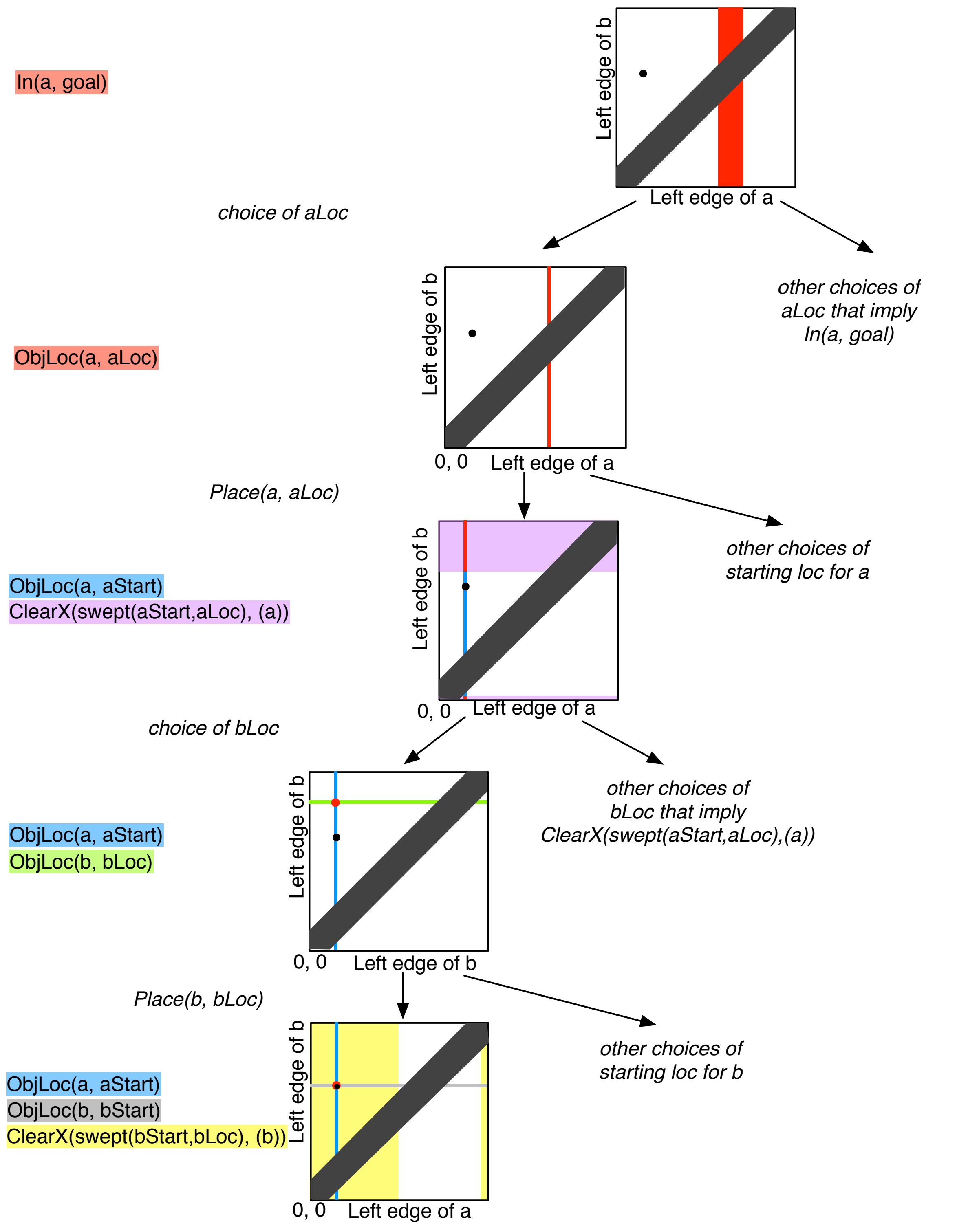


Figure 8: A path through the regression search tree in configuration space for the simple one dimensional environment.

- The root of the tree is the starting state of the regression search. It is the goal condition $\mathit{In}(a, \mathit{goal})$. That logical condition corresponds to a region of configuration space illustrated in red. The goal does not specify the location of b, so it can have any legal value; but the left edge of a is constrained to lie in an interval that will guarantee that the entire volume of a is in the goal region. For reference in all of the configuration-space pictures in this figure, the starting state, which is a specific position of both boxes, is shown as a black dot.

- The first choice in the search process is to choose a location for a that will cause it to be contained in the goal region. A *generator* will construct or sample possible locations for a; we illustrate a specific choice *aLoc*, shown in figure 6 as a short vertical blue line. The logical goal $\mathit{In}(a, \mathit{goal})$ is reduced to $\mathit{ObjLoc}(a, \mathit{aLoc})$. This is not the entire pre-image of the goal, and for completeness it might be necessary to backtrack and search over different choices.

- The next step in the search is to compute the pre-image of $\mathit{ObjLoc}(a, \mathit{aLoc})$ under the action $\textsc{Place}(a, \mathit{aLoc})$. In order for that action to be successful, it must be that a is located at some initial location. The Place operator allows the starting location of the object to be chosen from among a set of options including that object's location in the initial state, as well as other salient locations in the domain. In this example, we choose to move it from its location in the initial state, *aStart*; to enable that move, we must guarantee that the region of space through which a moves as it goes from *aStart* to *aLoc* is clear of other objects. The requirement that a be located at *aStart* corresponds to the vertical blue locus of points in the configuration space. The requirement that the swept volume be clear turns into a constraint on the location of b. If the left edge of b is in one of the intervals shown in purple, then it will be out of the way of moving a. The pre-image is the conjunction of these two conditions, which is the intersection of the blue and purple regions in the configuration-space figure, which is shown in red.

- The current configuration of the objects (black dot) is not in the goal region, so the search continues. We search for a placement of object b that will cause it to satisfy the *ClearX* constraint, suggesting multiple positions including *bLoc*. This leads to the configuration-space picture in which the green line corresponds to object b being at location *bLoc*. Intersected with the constraint, in blue, that a be at location *aLoc*, this yields the subgoal of being in the configuration indicated by the red dot.

- Finally, the operator $\textsc{Place}(b, \mathit{bLoc})$ can be used to move object b, subject to the preconditions that b be located at *bStart* (shown as the gray stripe) and that the swept volume for b from *bStart* to *bLoc* be clear except for object b. The clear condition is a constraint on object a to be in the yellow region. The intersection of all these conditions is a point which is equal to the starting configuration, and so the search terminates. (Note that, in general, the pre-image will not shrink to a single point, and the search will terminate when the starting state is contained in a pre-image).

The result of this process is a plan, read from the leaf to the root of the tree, to move object b to location *bLoc* and then move object a to location *aLoc*, thereby achieving the goal.

3.3 Hierarchy

Viewed as motion planning problems, the tasks that we are interested in solving have very high dimensionality, since there are many movable objects, and long planning horizons, since the number of primitive actions (such as linear displacements) required for a solution is very large. They are also *multi-modal* [Siméon et al., 2004, Hauser and Ng-Thow-Hing, 2011, Barry et al., 2012] requiring choosing grasps and finding locations to place objects, as well as finding collision-free paths. As such, they are well beyond the state of the art in motion planning.

Viewed as task-planning problems, the tasks that we are interested in solving require plans with lengths in the hundreds of actions (such as pick, place, move, cook). The combination of long horizon, high forward branching factor, and expensive checking of geometric conditions means that these problems are beyond the state of the art for classic domain-independent task-planning systems.

Our approach will be to use hierarchy to reduce both the horizon and the dimensionality of the problems.

- We reduce the horizon with a temporal hierarchy in terms of abstract actions, which comprise the process of achieving conditions to enable a primitive action as well as the primitive action itself. The lowest layer of actions are hand-built primitive actions, and the upper layers are constructed dynamically by a process of postponing pre-conditions of actions.
- The hierarchy also reduces dimensionality of sub-problems because fewer objects and fewer properties of those objects are relevant to sub-problems; the factored nature of the logical representation, which allows us to mention only those aspects of state that are relevant, naturally and automatically selects a space to work in that has the smallest dimensionality that expresses the necessary distinctions.

We have developed a hierarchical interleaved planning and execution architecture. In HPN, a top-level plan is made using an abstract model of the robot's capabilities, in which the "primitive" actions correspond to the execution of somewhat less abstract plans. The first subgoal in the abstract plan is then planned for in a less abstract model, and so on, until an actual primitive action is reached. Primitive actions are executed, then the next pending subgoal is planned for and executed, and so on. This process results in a depth-first traversal of a planning and execution tree, in which detailed planning for future subgoals is not completed until earlier parts of the plan have both been constructed and executed. It is this last property that we refer to as "in the now."

This mechanism results in a natural divide-and-conquer approach with the potential for substantially decreasing complexity of the planning problem, and handles unexpected effects of execution through monitoring and replanning in a localized and efficient way. On the other hand, a consequence of decomposition is that highly coupled problem instances will generally not be handled efficiently (but, there is no method that can handle all problems efficiently). We believe this trade-off is worth making since we expect that highly coupled problems are relatively rare in common-place activities. It also requires care and insight to construct a hierarchical domain specification that works effectively with HPN.

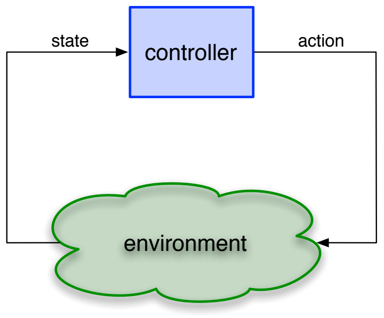


Figure 9: In a Markov decision process, the state of the environment is completely observable, and the job of the controller is to map environment states into actions.

4 HPN in uncertain domains

In this section we describe an approach to applying HPN in domains with uncertainty both about the outcomes of actions and about the current state of the world.

4.1 Uncertain outcomes

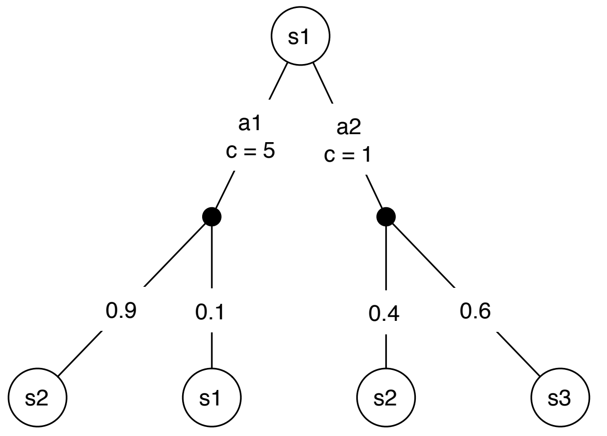
In a Markov decision process, the state of the environment is completely observable, and the job of the controller is to map environment states into actions. Our MHPN algorithm provides a planning and execution method that serves as a controller for an MDP, as shown in figure 9.

Our approach to handling probabilistic uncertainty in the outcomes of actions is to: construct a deterministic approximation of the domain, plan a path from the current state to a state satisfying the goal condition; execute the first step of the plan; observe the resulting state; plan a new path to the goal; execute the first step; etc. We will formulate the problem of planning in the deterministic approximation as the problem of finding a minimum cost path in a graph with positive weights.

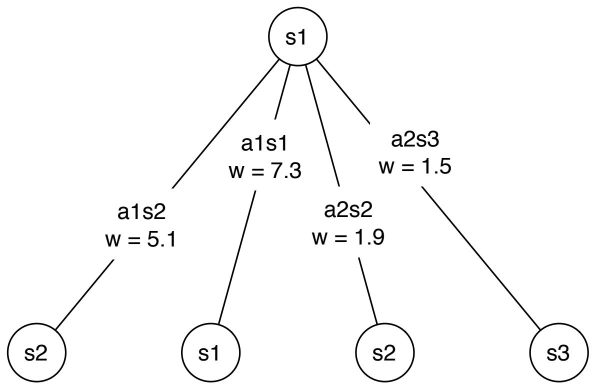
4.1.1 Determinizing the model

There are several potential strategies for constructing a determinized model. A popular approach is to assume, for the purposes of planning, that the most likely outcome is the one that will actually occur. This method can never take advantage of a less-likely outcome of an action, even if it is the only way to achieve a goal. However, there are many domains in which success hinges on, at some point, obtaining an outcome that is not the most likely: consider a domain in which a robot must repeatedly try an action that has a probability of 0.6 of failing: in expectation, it only takes 2.5 tries to succeed, but success hinges on an outcome other than the most likely one actually occurring. It is important to be able to solve problems of this type, so we begin by showing how to convert a Markov decision process into a weighted graph.

Because we are interested in regression-based planning using fluent-based representations of sets of world states, we wish to retain the basic structure of planning to achieve a goal, rather than optimizing the sum of state and action costs over a fixed finite horizon or over the infinite horizon. These problem formulations are mostly inter-convertible [Barry et al., 2011, Bertsekas and Tsitsiklis, 1996], but for problems that are naturally articulated in terms of reaching a desired state,



(a) Search tree for stochastic shortest path problem. First layer is the choice of action, with a fixed cost of 5 for a_1 and of 1 for a_2. Second layer is a distribution over outcomes for each action.



(b) Search tree for derived deterministic model (CLDSSPP), in which there is an outgoing arc for each action/outcome pair, with associated weight equal to $c - \log p$ where c was the cost associated with the action and p the probability of the transition.

Figure 10: Probabilistic search tree and its deterministic approximation.

it is more efficient to stay within that representation, and so we focus on stochastic shortest-path problems.

Definition 1. *A* Markov decision process*(*MDP*) is a tuple $\langle S, A, T, R\rangle$ where S is a set of world states, A is a set of actions, T is a probabilistic Markov transition model, with $T(s, a, s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$, and R is a reward function where $R(s, a)$ is the immediate value of taking action a in state s.*

Definition 2. *A* stochastic shortest-path problem*(*SSPP*) is an* MDP *in which all rewards are negative, there is a set $G \subset S$ of goal states, and the objective is minimize the total expected cost (negative reward) incurred before reaching a state in G and terminating. We define cost function $C(s, a)$ in the* SSPP *to be $-R(s, a)$ in the original* MDP.

We will show how to consider all possible outcomes in a SSPP, but rather than modeling the outcome as a randomized choice that is made by nature, instead modeling it as a choice that can be made by the agent [Barry et al., 2011, Blum and Langford, 1999]. We will convert an SSPP into a *determinized* SSPP as follows.

Definition 3. *A* determinized SSPP *is a tuple $\langle S, A', W, G\rangle$ where: $\langle S, A, T, C, G\rangle$ is a* SSPP*; S is a set of states which are nodes in a graph; A' is a set of actions (a, s'), so that $(s, a, s') \in S \times A \times S$ is a directed arc from node s to node s'; and W is a weight function, so that $W(s, a, s')$ is the weight on arc (s, a, s'), which may be infinite.*

There are different ways of defining the weight function W depending on the transition model T and costs C of the original problem.

Definition 4. *A* transition-likelihood DSSPP *(*TLDSSPP*) is a* DSSPP *where $W(s, a, s') = -\log T(s, a, s')$.*

Proposition 1. *The least-cost path from a state s to some state $s' \in G$ in a* TLDSSPP *is the path through the original* SSPP *with the highest likelihood of reaching G from s.*

Proof. The most likely path is a sequence of states $s_1, a_1, \ldots, a_{n-1}, s_n$ where $s = s_1$ and $s_n \in G$, that maximizes $\prod_{i=1}^{n-1} T(s_{i+1} \mid s_i, a_i)$. This is equivalent to minimizing $\sum_{i=1}^{n-1} -\log T(s_{i+1} \mid s_i, a_i)$, which will be the least-cost path through the TLDSSPP. □

Generally, we would like to find paths that are both likely to reach a goal state and that also minimize transition cost incurred along the way. There is not an immediately obvious way to combine cost and likelihood to get a determinized model. We use a parameterized combination, defined below.

Definition 5. *An* α-cost-likelihood DSSPP *(*CLDSSPP*) is a* DSSPP *where*

$$W(s, a, s') = \alpha C(s, a) - \log T(s, a, s') \enspace .$$

Figure 10(a) shows the first two layers of a standard search tree for a SSPP: at the top level is a choice between actions a_1 and a_2, with a fixed cost of 5 for a_1 and 1 for a_2. Then, for each action, there is a probabilistic branch on outcomes. Figure 10(b) shows the search tree for the corresponding CLDSSPP, in which any of the outcomes of each action can be selected, each with a weight equal to $c - \log p$ where c is the cost of the action and p the probability of the selected outcome given the start state and action. Note that there are now two arcs from s_1 to s_2, one with with high probability (0.9) and high cost (5), resulting in a weight of 5.1 and one with lower probability (0.4) and lower cost (1), resulting in a weight of 1.9.

4.1.2 Regression with costs

In the HPN framework, we need to be able to perform regression search, chaining pre-images backward from the goal to eventually reach a set of states that contains the initial state. This process is a relatively straightforward search through the space of *subsets* of the state space. We will ultimately use logical expressions to compactly denote large or infinite subsets of the state space, so the complexity of this search approach may be independent of the size of the underlying state space.

In the following, we show how to convert a cost-likelihood DSSPP into a regression cost problem RCP, which is also a weighted graph, but one in which the nodes are sets of states in the original problem and arcs are labeled by actions and costs.

Definition 6. *Given a* DSSPP $\langle S, A, W, G\rangle$ *we define a* regression cost problem *(*RCP*) to be a tuple* $\langle N, A, W'\rangle$*, where* N *is a set of pre-images as defined below,* A *is as in the* DSSPP *and* W' *are the weights for transitions among pre-images.*

Define the weight-w pre-image *of* $n \in 2^S$ *under action* $a \in A$ *to be the set of states that have a weight* w *arc leading to some state* s' *in* n *via action* a*:*

$$I(n, a, w) = \{s \mid \exists s' \in n.\, W(s, a, s') = w\} \enspace .$$

The set N *of pre-images is constructed recursively starting from the goal set* G *of the original* DSSPP*:*

- G *is an element of* N*;*
- *For any* $n \in N$*,* $a \in A$*, and* w *such that* $I(n, a, w)$ *is non-empty,* $I(n, a, w)$ *is an element of* N*, and* $W'(I(n, a, w), a, n) = w$*.*

We will formulate our search problems as regression cost problems, with the goal of finding the least-cost path through the graph to a node that contains the initial state of the original problem. When we use logical operator descriptions to characterize a transition model for planning, we are encoding the RCP directly.

4.1.3 Markov HPN

We extend the basic HPN algorithm to apply to domains with outcome uncertainty by retaining the same depth-first planning and execution architecture, but additionally monitoring the effects of actions in an effort to ensure that the action being currently selected is the first step in a plan that has positive probability of achieving the goal, and that extra, unnecessary actions are not taken.

Letting s_{now} be the current world state, γ be a logical description of the set of goal states, α be an abstraction level, and *world* be an interface to a real or simulated robot system, we define *mhpn* as follows:

```
MHPN($s_{now}, \gamma, \alpha, world$):
    $p = \text{PLAN}(s_{now}, \gamma, \alpha)$
    while $s_{now} \in envelope(p)$
        $i = \arg\max_i s_{now} \in g_i(p)$
        if ISPRIM($\omega_i(p)$)
            $s_{now} = world.\text{EXECUTE}(\omega_i(p))$
        else
            $s_{now} = \text{MHPN}(s_{now}, g_i(p), \text{NEXTLEVEL}(\alpha, \omega_i(p)), world)$
    return $s_{now}$
```

MHPN begins by constructing a plan, which is the solution to a regression cost problem (to reduce clutter in the definition, the domain dynamics are not explicitly passed in). PLAN returns a list, $((-, g_0), (\omega_1, g_1), ..., (\omega_n, g_n))$ where the ω_i are operator instances, $g_n = \gamma$, g_i is the pre-image of g_{i+1} under ω_i, and $s_{now} \in g_0$.

We define the *envelope* of the plan to be the union of the pre-images of all the steps in the plan:

$$envelope(p) = \bigcup_{i=0}^{n-1} g_i(p) \quad .$$

As long as the current state is in the envelope of the plan, then the plan can be executed, with positive probability of resulting in a state that satisfies the goal. If the execution of a plan step fails, but the state remains in the envelope of the plan, then it is reasonable to continue executing this plan from the appropriate point. Similarly, if a serendipitous event occurs, moving the state "forward" in the plan, it is also reasonable to continue executing the plan from the appropriate plan step. If, however, the state exits the region over which the plan is expected to work, then the call to MHPN returns, which will trigger a decision either to replan for this subgoal, to execute a different plan step at the next higher level of abstraction, or to return to a yet higher level of MHPN.

To select the plan step to execute, we find the preimage g_i with the highest index i such that $s_{now} \in g_i$. This is the plan step that is closest to the end of the plan such that, were we to begin plan execution from that step, a state satisfying the goal condition could come to hold. This is

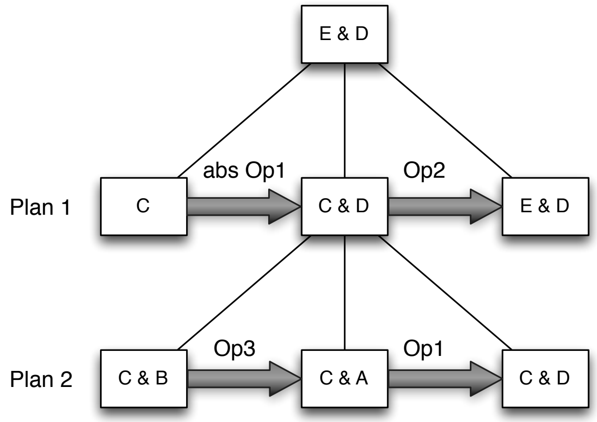


Figure 11: MHPN execution example

very much like the execution rule, in triangle tables, of executing the highest true kernel [Nilsson, 1985].

MHPN deals with pre-images represented as sets of fluents and it only needs to test whether the current, fully specified, state satisfies these fluents. The fluents are defined in part by a test function that determines their truth value in a state.

To ensure that execution is persistent at the highest level, and to handle passing in the highest abstraction level initially, we define MHPNTOP, which is the top-level call for MHPN.

```
MHPNTOP($s_{now}, \gamma, world$):
    **while** $s_{now} \notin \gamma$
        MHPN($s_{now}, \gamma, \alpha_0, world$)
```

Figure 11 illustrates the MHPN execution process. Assume we have a domain with the following simplified operators:

$$\begin{aligned} abstract\,Op1 &: \{\,\} \rightarrow D \\ Op1 &: A \rightarrow D \\ Op2 &: C \rightarrow E \\ Op3 &: B \rightarrow A \end{aligned}$$

The letters on the left side of the arrow are preconditions, and those on the right are effects. Assume that we have an abstract version of *Op1* that has postponed the precondition A. First, we construct plan 1, which consists of abstract *Op1* followed by *Op2*. Assuming that condition C is true in the world (otherwise, the planning process would not have terminated) but that D is not (otherwise, it would have generated only a one-step plan), HPN finds the rightmost true pre-image, which is C and executes the associated operations. In this case, it is an abstract version of *Op1*, so HPN is called recursively with $\gamma = C \;\&\; D$ with abstraction value for *Op1* incremented. Now, plan 2 is constructed, which consists of *Op3* followed by *Op1*. Assuming that B and C are true in the world, but A is not, we would select *Op3* for execution. Now, we can consider three different possible outcomes.

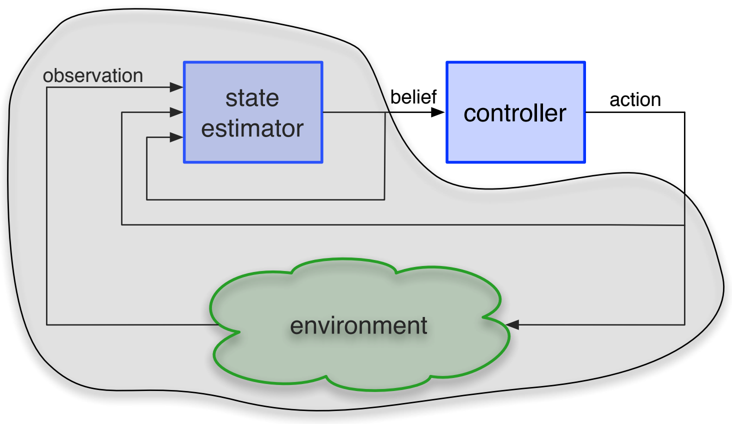


Figure 12: From the perspective of the controller in a POMDP, the belief-state estimation module becomes part of the external environment, and the job is to map belief states into actions in such a way as to drive the belief state into a desired set.

- The expected outcome is that C remains true and A becomes true, in which case we would go on to execute the primitive *Op1*.
- Another possible outcome could be that C remains true and D becomes true. In this case, γ for the recursive call to HPN has become true, and so control would return up a level. We would find that C & D is true but E & D is not, and so execute *Op2*.
- An additional possibility is that C becomes false. In this case, we find that we are no longer in the envelope of plan 2, and so the recursive call to HPN would return. Now, we would also find that we are no longer in the envelope of plan 1. Control would return to the top level, and a new plan would be constructed for achieving E & D.

This last case demonstrates that, even though the execution monitoring in HPN is localized to the plan currently being executed, the conditions that support the global usefulness of that plan are passed down in such a way that execution will terminate if those conditions go false. Notice that the condition C is only critical for the last step of the high-level plan. But it is carried along through the recursive call for the first step of the high-level plan, so that as soon as it goes false, the recursion will return to level in the planning and execution process that does not depend on C.

We can see the entire planning and execution strategy as a closed-loop feedback controller, which selects its next action contingent on the result of the previous action and seeks always to move closer to a state in the goal set. For deterministic environments, Kaelbling and Lozano-Pérez [2012] states conditions on the domain and on the hierarchy of planning models that guarantee completeness of HPN in the sense that it will eventually reach a goal state if one was reachable from the initial state. We conjecture that, in the probabilistic case, as long as each plan that is constructed has a non-zero probability of success, then eventually a goal state will be reached.

4.2 Uncertain state

When there is uncertainty about the current state of the world, we can formalize the problem as a POMDP. If we decompose the problem as shown in figure 3, we can separate the problems of state estimation and control. Assuming a state estimator is already fixed, the control problem can be seen as shown in figure 12. The problem for the controller is to map belief states into actions: from this perspective, the "environment" now encompasses both the external world and the belief state estimator. The planning and execution system must map belief states into actions in such a way as to drive the belief state into some desired set of belief states. The belief state dynamics can be described as a continuous-state Markov decision process. We make small extensions to the MHPN algorithm to apply it to belief space.

As simple example, consider a mobile robot that is uncertain about its discrete location in an environment made up of rooms and hallways. Its belief state can be represented by a multinomial distribution over the discrete locations, specified with a probability value for each location. The goal might be for the robot to believe with probability greater than 0.9 that it is in location 3. In order to select actions that will achieve this goal, the robot has to consider their effects on its belief: moving in the world will tend to move the probability mass in the distribution, possibly blurring the distribution in the process; taking sensing actions will tend to sharpen the distribution. Planning in belief space supports selecting an appropriate combination of moving and sensing actions to drive the belief state into the set of beliefs that satisfy the goal condition.

4.2.1 HPN **in belief space**

Planning in belief space is generally quite complex, because it seems to require representing and searching for trajectories in a very high-dimensional continuous space of probability distributions over world states. This is analogous to the problem of finding plans in very high-dimensional continuous space of configurations of a robot and many objects. We take direct advantage of this analogy and use logical fluents to specify limited properties of belief states, as our earlier HPN approach does for properties of geometric configurations. So, for instance, we might characterize a set of belief states by specifying that "the probability that the cup is in the cupboard is greater than 0.95." Pre-image backchaining allows the construction of high-level plans to achieve goals articulated in terms of those fluents; it will work effectively when the pre-images can also be described using relatively small sets of these fluents.

The problem of updating the state estimate based on an action and observation comes up in two distinct ways in our approach: first, in the *state estimation* module as shown in figure 3 and second as part of the model of the dynamics of the belief state that is used during planning (we have to predict how taking an action will change the belief state). We have found that it is important that the belief-state representation and update in the state estimation module be done was accurately as possible: that belief state is our only proxy for the entire history of actions taken and observations made by the robot. However, in the planning process, the model can be considerably more approximate: errors made in planning due to model approximation can typically be corrected by observing a new resulting state and selecting actions appropriate to it.

The basic planning and execution strategy for HPN need not be changed for planning in belief space. Whereas, in the case of solving an MDP we reduced the planning problem to a regression cost problem where the nodes represented sets of world states, now we will address the planning problem as a regression cost problem where the nodes represent sets of *belief states*. We then extend the recursive MHPN planning and execution method to the BHPN method, shown below, where we substitute a *belief state*, b_{now}, for the world state and add an update of the belief state based on an observation resulting from executing the action in the world:

```
BHPN($b_{now}, \gamma, \alpha, world$):
    $p = $ PLAN$(b_{now}, \gamma, \alpha)$
    **while** $b_{now} \in envelope(p)$
        $i = \arg\max_i b_{now} \in g_i(p)$
        **if** ISPRIM$(\omega_i(p))$
            $obs = world.$EXECUTE$(\omega_i(p))$
            $b_{now} = belief.$UPDATE$(\omega_i(p), obs)$
        **else**
            $b_{now} = $ BHPN$(b_{now}, g_i(p), $ NEXTLEVEL$(\alpha, \omega_i(p)), world)$
    **return** $b_{now}$

BHPNTOP($belief, goal, world$):
    **while** $belief \notin goal$
        BHPN$(belief, goal, \alpha_0, world)$
```

After each primitive action (which may in fact involve calling a planner, and following the resulting trajectory, or executing a guarded move or other control loop) is executed, an observation is made in the world and the belief state is updated to reflect both the predicted transition and the information contained in the observation *obs*. Given an action and an observation, the belief state update is deterministic. However, the particular observation that will result from taking an action in a state is probabilistic; that uncertainty is handled by the BHPN structure in the same way that uncertainty of action outcomes in the world was handled in the MHPN structure, by planning in a determinized model, monitoring execution, and replanning when the plan is invalidated.

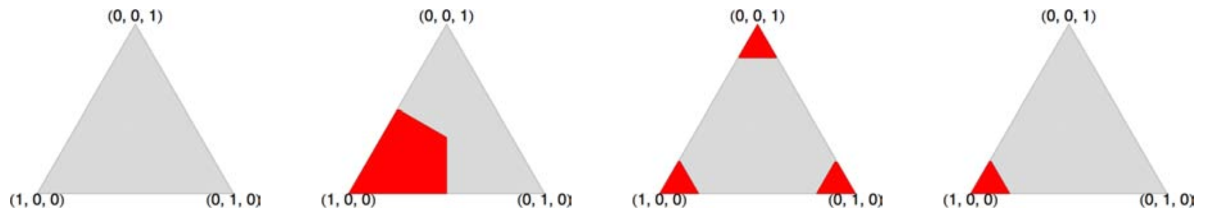
5 Logical characterization of beliefs for planning

Our strategy for creating a controller for a POMDP will be to provide a description of the *belief process*: that is, the way that the belief state evolves given actions taken by the robot. Because we are interested in very large domains, we use abstractions afforded by geometric and logical representations to compactly represent the belief process dynamics. This section describes a method for representing sets of belief states and their dynamics using logical fluents and operator descriptions, and shows how they can be used in regression-based planning and embedded into the BHPN recursive planning and execution architecture.

When planning in belief space, goals must also be described in belief space. Example goals might be "With probability greater than 0.95, the cup is in the cupboard." or "The probability that more than 1% of the floor is dirty is less than 0.01." These goals describe *sets* of belief states. The process of planning with pre-image backchaining computes pre-images of goals, which are themselves sets of belief states. Our representational problem is to find a compact yet sufficiently accurate way of describing goals and their pre-images.

5.1 Discrete-state domain

We begin by considering a very simple, discrete-state domain, in which there is a single object that can be in one of three possible locations. We will use this domain to illustrate the use of logical fluents to characterize beliefs, to show how operator descriptions can be used to specify approximate



(a) 2D simplex

(b) Subset of belief states in which state 0 is most likely. These belief states satisfy $MLLoc(l_0)$.

(c) Subset of belief states in which one of the states is believed to be true with probability $\geq$ 0.2. These belief states satisfy $BVLoc(0.2)$.

(d) Subset of belief states in which state 0 is believed to be true with probability $\geq$ 0.2. Note that this is the intersection of the previous two sets. These belief states satisfy $BLoc(l_0, 0.2)$.

Figure 13: Two-dimensional simplex represents the set of all possible belief states over a discrete state space with three states.

deterministic dynamics in stochastic domains, and to demonstrate the results of planning and execution.

5.1.1 Belief states

In general, in a discrete-state domain with n states, a belief state is a binomial distribution over those states, specified as an n-dimensional vector $b = \langle p_0, \ldots, p_{n-1} \rangle$, with $0 \leq p_i \leq 1$ and $\sum_{i=0}^{n-1} p_i = 1$. The belief b is a point in the $n-1$-dimensional unit simplex; the dimension is $n-1$ rather than n because the constraint that the elements of b sum to 1 reduces the degrees of freedom by 1. So, in a domain whose state is represented by random variable S, taking on 3 discrete values, s_0, s_1, s_2, representing locations of an object, a belief state is characterized by $b = \langle \Pr(S = s_0), \Pr(S = s_1), \Pr(S = s_2) \rangle$, which is a point in a triangle, as shown in figure 13(a). We will use $\Pr_b(e)$ to denote the probability assigned to event e (a subset of possible worlds) by the distribution b.

5.1.2 Logical language and interpretation

In this section we informally define a logical language and its desired interpretation; this language is used to specify goals for the planning system and to describe the domain dynamics using operator descriptions.

Logical statements are made up of conjunctions of fluents. We use these logical statements to specify conditions on belief states of the robot. The meaning of a fluent f is defined by specifying a test, $\tau_f : \mathit{args}, b \rightarrow \{\mathit{true}, \mathit{false}\}$, where args are the arguments of fluent f, b is a belief state, and the fluent $f(\mathit{args})$ holds in belief state b iff $\tau_f(\mathit{args}, b) = \mathit{true}$. A ground fluent (all of whose arguments are constants) then denotes a subset of all possible belief states; a conjunction of fluents denotes a set of belief states that is the intersection of the belief sets denoted by the elements of the conjunction.

In the simple discrete-state domain, we define fluents that describe sets of belief states that are useful for characterizing the domain dynamics. Recall that S is a random variable denoting the state of the world which, in this example, is just the location of the object.

- *MLLoc*(l): location l is the *most likely* location of the object. The corresponding test is

$$\tau_{MLLoc}((l),b) := \forall l'.\ \Pr_b(S=l) \geq \Pr_b(S=l')\ \ .$$

- *BLoc*(l,ϵ): the object is *believed to be located* in location l with probability at least $1-\epsilon$. The corresponding test is

$$\tau_{BLoc}((l,\epsilon),b) := \Pr_b(S=l) \geq 1-\epsilon\ \ .$$

- *BVLoc*(ϵ): we *believe the value of the location* of the object with probability greater than $1-\epsilon$. Note that this fluent does not commit to which location the object is in. Such fluents are useful in characterizing the future effects of information-gathering actions: we cannot say, now, which location we will believe contains the object, but we can say that, in the future, we will have high confidence that it is in one of the locations. The corresponding test is

$$\tau_{BVLoc}((\epsilon),b) := \exists l.\ \Pr_b(S=l) \geq 1-\epsilon\ \ .$$

The sets of belief states, in the three-location domain, corresponding to each of these fluents are shown in figures 13(b), 13(c), and 13(d). It is interesting to see that $BLoc(l,\epsilon) \equiv MLLoc(l)\ \&\ BVLoc(\epsilon)$; that is, that the set of belief states characterized by $BLoc(l,\epsilon)$ is equal to the intersection of the sets of belief states characterized by $MLLoc(l)$ and by $BVLoc(\epsilon)$.

Because we are working in an infinite domain, in addition to specifying tests for the fluents, we must specify methods that determine whether two fluents are in contradiction and whether one entails another [Kaelbling and Lozano-Pérez, 2012]. The methods for this domain are provided in appendix A.1.

5.1.3 Belief state dynamics

We can use these fluents to characterize the effects of the robot's actions on the belief state. We continue with our simple example domain, adding two actions: moving an object from one location to another, and looking in a particular location to see if the object is there.

Move When the robot attempts to move an object from location l_i to location l_j, then the object will end up in location l_j with probability $1-p_{fail}$ if, in fact, the object was in location l_i to being with; otherwise, it will remain in its original location. How can we characterize the effect of this action on the belief?

Let $b_i = \Pr_b(S=l_i)$. Then, although the outcome of this action in the world is probabilistic, its effect on the belief state is deterministic:

$$\begin{aligned}\Pr_{b'}(S=l_j) &= b'_j \\ &= b_i(1-p_{fail}) + b_j\ \ ,\end{aligned}$$

where b' is the belief state that results from taking the $\textsc{move}(l_i,l_j)$ action in belief state b.

We can describe this action using an operator description suitable for regression-based planning:

```
MOVE($l_{start}, l_{target}$):
    effect: $BLoc(l_{target}, \epsilon)$
    choose: $l_{start} \in Locations \setminus \{l_{target}\}$
    pre: $\epsilon \geq p_{fail}$
        $BLoc(l_{start}, moveRegress(\epsilon))$
    prim: MOVEPRIMITIVE($l_{start}, l_{target}$)
    cost: 1
```

The variables l_{start} and l_{target} denote the initial and final locations of the object. Recall that **choose:** means that there is an instance of this operator schema for each possible binding of the choice variable. We let *Locations* indicate the universe of possible object locations in the domain. Because the result is deterministic, the cost is just the base cost of executing the action (in this paper, all such costs are 1; it remains for us to develop a more general treatment of costs in hierarchical problems).

The *moveRegress* function determines the minimum confidence required in the location of the object on the previous step, in order to guarantee confidence ϵ in its location at the resulting step, even if the initial confidence is zero. Derivations supporting propositions made in the body of the paper appear in appendix B.

Proposition 2.

$$moveRegress(\epsilon) = \frac{\epsilon - p_{fail}}{1 - p_{fail}}$$

Note that if $\epsilon < p_{fail}$ then the operator cannot succeed.

Look When the robot looks in a location l_i, it may either observe the object there, or not, governed by the following probabilities:

$$\begin{aligned} \Pr(O = true \mid S = l_i, A = \text{LOOK}(l_i)) &= 1 - p_{fn} \\ \Pr(O = true \mid S \neq l_i, A = \text{LOOK}(l_i)) &= p_{fp} \end{aligned}$$

where p_{fn} is the *false negative* probability of not seeing the object when it is really there, and p_{fp} is the *false positive* probability of seeing the object when it is not there.

The effects of an observation action on the belief state are non-deterministic in this case, because the result depends on which observation is received. There are four cases of interest: how b_i changes

1. When the robot looks at location l_i and does see the object;
2. When the robot looks at location l_i and does not see the object;
3. When the robot looks at location $l_j \neq l_i$ and does see the object; and
4. When the robot looks at location $l_j \neq l_i$ and does not see the object.

In cases 2 and 3, b_i will decrease; in cases 1 and 4, it will increase. We do not need to characterize the complete dynamics of the belief state: it is sufficient to provide a set of operations that can be used to reach any desired goal. We expect to have goals that require the concentration of the probability mass within the belief, so we only go on to formalize the effects of cases 1 and 4.

The effects on $\Pr(S = l_{target})$ when the action is to look at location l_{target} and the object is detected are characterized by this operator description:

LookToVerify(l_{target}):

effect: $BLoc(l_{target}, \epsilon)$
pre: $BLoc(l_{target}, lookPosRegress(\epsilon))$
prim: LookPrimitive(l_{target})
cost: $1 - \log(posObsProb(lookPosRegress(\epsilon)))$

It is important to note that we are describing the combined effects of taking the action in the world, receiving an observation, and performing the belief-state update.

Proposition 3.

$$lookPosRegress(\epsilon) = \frac{\epsilon(1 - p_{fn})}{\epsilon(1 - p_{fn}) + p_{fp}(1 - \epsilon)} \quad .$$

If there is a probability ϵ_n that the object is not in the location being observed, then the probability of getting a positive observation is

$$posObsProb(\epsilon_n) = (1 - p_{fn})(1 - \epsilon_n) + p_{fp}\epsilon_n \quad :$$

the first term is the probability that the object is actually there, times the probability it will be observed if it is there; the second term is the probability that the object is not there times the probability that a false positive observation of it will be made, nonetheless. Thus, the cost of this operator is 1 for taking the action plus $-\log posObsProb(\epsilon_n)$ to account for the likelihood of failure. Note, though, that this is a bound on the actual cost: at the time the operator is applied, in order for the precondition to be satisfied, the belief that the object is in l_{target} must be *at least* $1 - lookPosRegress(\epsilon)$; if the belief is greater than that, then the precondition is still satisfied, but the success probability will be higher and, thus, the actual expected cost lower.

With no additional operators, problems in this domain can be solved by looking to confirm that an object is in an expected or desired location. To enable other strategies, we add an operator that characterizes the effect of looking at a location l_{target} in order to rule it out (case 4), increasing the likelihood of $l_{interest}$:

LookToRuleOut($l_{target}, l_{interest}$):

effect: $BLoc(l_{interest}, \epsilon_{interest})$
choose: $l_{target} \in Locations \setminus \{l_{interest}\}$
pre: $BLoc(l_{target}, \epsilon_{target})$
$BLoc(l_{interest}, lookNegRegress(\epsilon_{interest}, \epsilon_{taret}))$
prim: LookPrimitive($l_{interest}$)
cost: $1 - \log(1 - posObsProb(\epsilon_{target}))$

Proposition 4.

$$lookNegRegress(\epsilon) = \frac{(1 - p_{fp}) - (1 - \epsilon_i)(p_{fn}(1 - \epsilon_j) + \epsilon_j(1 - p_{fp}))}{1 - p_{fp}} \quad .$$

The cost of this operator is 1 for taking the action plus $-\log(1 - posObsProb(\epsilon_j))$ to account for the likelihood of failure (remember that this action "succeeds" in increasing the likelihood of $l_{interest}$ when it looks at location l_{target} and *fails* to see the object there. This action will be selected when, for example, there is a nearby location with a relatively high probability, so that it is cheaper to move to and observe that location to rule it out than it is to move to the more likely but more distant location.

5.1.4 Execution example

In this section, we walk through the planning and execution process for the goal $BLoc(l_0, 0.05)$, which means that the robot wants to come to believe, with probability greater than 0.95, that the object is in location l_0. The initial belief state is $(0.3, 0.2, 0.5)$; that is, the robot believes the object is in location l_0 with probability 0.3, in location l_1 with probability 0.2 and in location l_2 with probability 0.5. The parameter values are: $p_{fail} = 0.2$, $p_{fp} = 0.1$, and $p_{fn} = 0.2$.

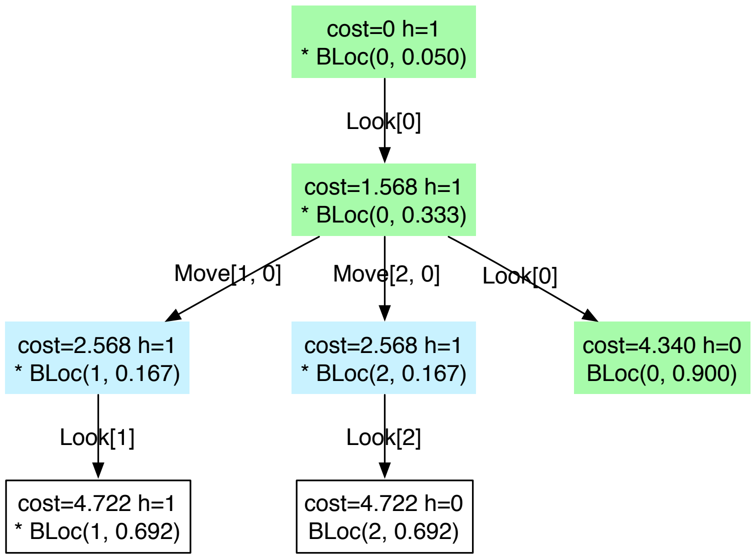
Figure 14(a) shows the search tree for the first planning problem. Nodes in green represent the solution path; nodes in blue have been expanded (their children added to the search queue), while nodes in white have not been expanded. Each node first shows the cost of the path from the root to the node and the heuristic value (we are using the very simple heuristic of the number of fluents that are not true in the goal state). The rest of the contents of a node is the list of fluents in the subgoal it represents; fluents with a star next to them are not true in b_{now}. The goal node is shown at the top.

There is only one way to achieve believing the object is in location 0 with high confidence, which is to look in location 0 to verify that the object is there. The move operation has sufficient error associated with it, that after a move, it will never be possible for the object's location to be known with error less than 0.05. The pre-image of the goal under a look operation is believing the object is in location 0 with error less than 0.33: this can be achieved by moving the object from any of the possible starting locations, or by looking again. It is instructive to look at the costs associated with these actions: the move operation has a deterministic effect in belief space and has a cost of 1, the look operation, whose pre-image is that the object is believed to be in location 0 with error 0.9 is very expensive (2.772) because it is fairly unlikely to actually see the object in location 0 from states in that pre-image. Searching further, we find that at least one look operation is necessary before moving, to establish that the object is in the starting location of the move, and taking those extra look actions into account, the (over) optimistic plan of looking twice in location 0 to verify that the object is there is found to be the best trade-off of cost and likelihood of success.

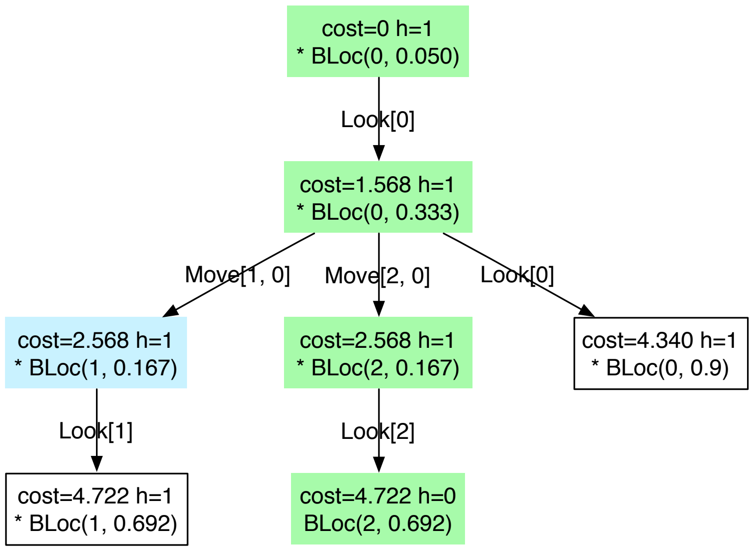
Figures 15 and 16 show the HPN planning and execution process for the three-location domain. At the beginning of each row is the relevant part of the HPN planning and execution tree: planning goals are blue, plan steps are pink, primitive actions are green, and indications of replanning steps are yellow and orange. In particular, when a plan step does not have the expected outcome, a yellow node is added to the tree indicating that it will be re-attempted. If the required antecedents for executing that step have become false, then an orange node indicating which antecedent is no longer true is added. At that point, the execution of that plan is abandoned and a new plan is constructed at the level above.

Following the tree is the pre-image sequence of the corresponding plan, drawn in the belief simplex. The red region is the goal, the orange region is its pre-image under the planned action, the yellow region is the pre-image of the orange, and the green (if any) is the pre-image of the yellow. The gray area is inside the belief simplex, but not in the union of the pre-images (the *envelope*) of the plan. Dots are belief states; the trajectory during execution is indicated by the arrows. The planning and execution proceeds as follows:

- Plan 1 corresponds to the path selected in figure 14(a), shown as the first two pink nodes in the figure 15(a), and as the colored pre-image sequence in the top row. The first step is executed, but the object is not observed to be in location 0. The belief state is updated and as we can see in figure 15(c), it has gone outside the pre-image of the plan, so replanning is

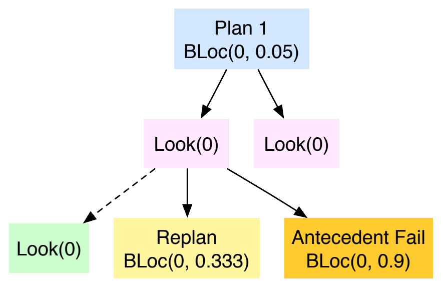


(a) Plan 1

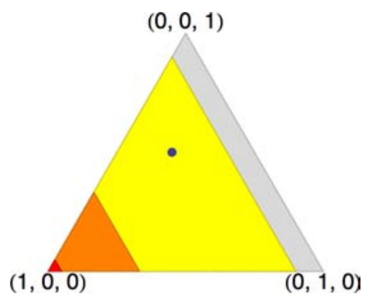


(b) Plan 2

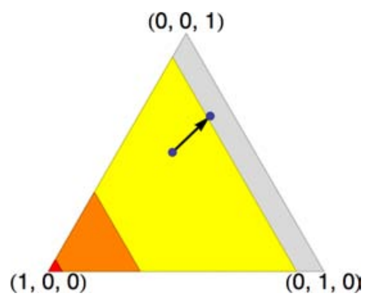
Figure 14: Search trees for the first two plans in the three-location domain. Note that these are not hierarchical planning and execution trees: they show the conventional A* search for a plan at a single level of abstraction. Because this is regression planning, the "root" node represents the goal condition.



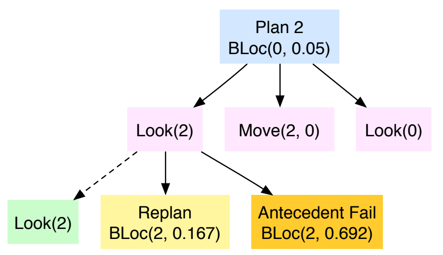
(a) Plan 1: Look(0), Look(0)



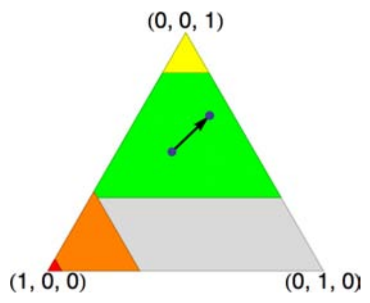
(b) Plan 1: Look(0), Look(0)



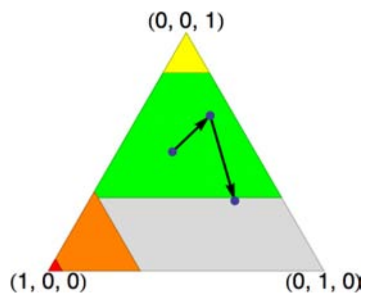
(c) Look does not see object in 0; belief state exits envelope of plan; replanning is triggered.



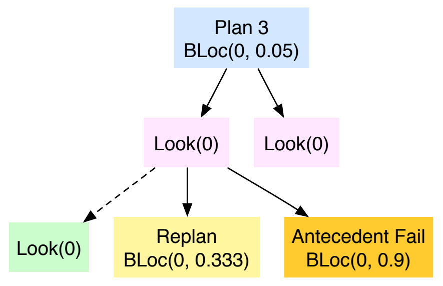
(d) Plan 2: Look(2), Move(2, 0), Look(0)



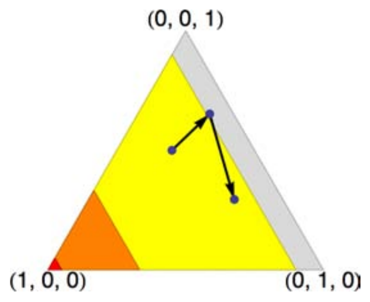
(e) Plan 2: Look(2), Move(2, 0), Look(0)



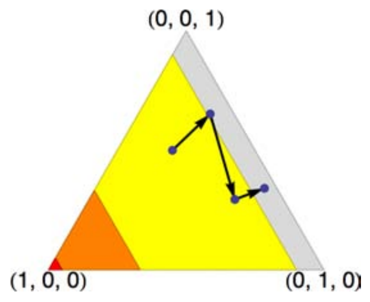
(f) Look does not see object in 2; belief state exits envelope of plan; replanning is triggered.



(g) Plan3: Look(0), Look(0)

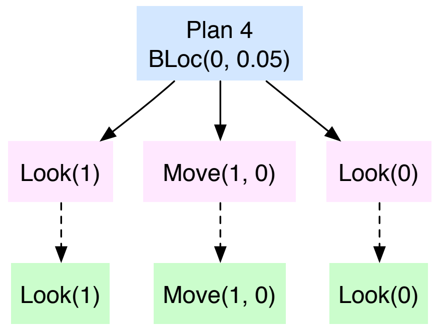


(h) Plan3: Look(0), Look(0)

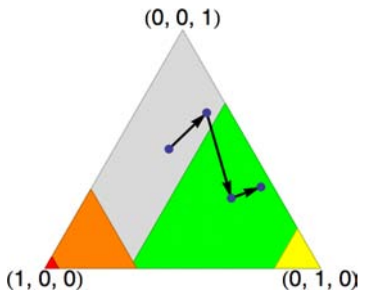


(i) Look does not see object in 0; belief state exits envelope of plan; replanning is triggered.

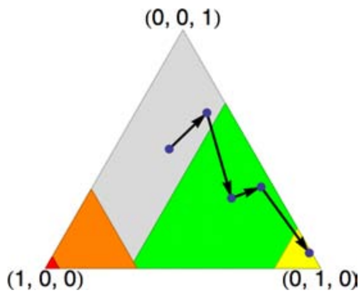
Figure 15: First three plans and execution steps in the three-location domain. Each row corresponds to a plan and its execution in the domain. Figure 15(b) shows the initial belief state as a dot, the goal set as a red triangle in the bottom-left corner, and the pre-images of two plan steps in orange and yellow. Figure 15(c) shows the trajectory in belief space that results from executing the first action: the new belief state is outside the envelope of the plan. Subsequent figures showing the belief simplex include the entire belief-space trajectory, starting from the initial belief state, and are described in more detail in the text.



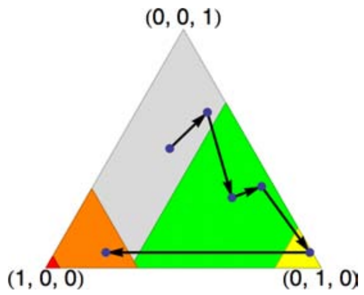
(a) Plan 4: Look(1), Move(1, 0), Look(0)



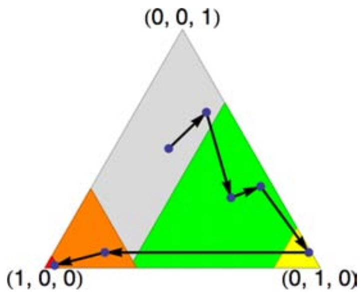
(b) Plan 4: Look(1), Move(1, 0), Look(0)



(c) Look sees object in 1.



(d) Move operation executed.



(e) Look sees object in 0.

Figure 16: The final plan and execution steps in the three-location domain.

triggered.

- Plan 2 (corresponding to the path in figure 15(e)) is shown in figure 15(d). It has as its first step to look in location 2, but when that step is executed, the object is not observed to be in location 1, so the belief state again exits the envelope of the plan (shown in figure 15(f)), and replanning is triggered.
- Plan 3, shown in figures 15(g) and 15(h), looks in location 0 but after execution, the robot does not see the object and replanning is triggered (figure 16(b)).
- Plan 4, shown in figures 16(a) and 16(b) looks in location 1 and sees the object (figure 16(c)), moves the object to location 1 (figure 16(d)), and finally looks in location 1 and sees the object (figure 16(d)).

A small set of discrete fluents can be used for effective planning in belief space.

5.2 Characterizing belief of a continuous variable

We can apply related techniques to characterize aspects of uncertainty in domains with continuous quantities, by requiring, for instance, that the mean of the distribution be within some value of the target and the variance be below some threshold. Generally, we would like to derive requirements on beliefs from requirements for action in the physical world. So, in order for a robot to move through a door, the estimated position of the door needs to be within a tolerance equal to the difference between the width of the robot and the width of the door. The variance of the robot's estimate of the door position is not the best measure of how likely the robot is to succeed, because if the robot's error is such that it cannot go through the door, it does not matter whether it was off by 1cm or 1 meter. Instead, we will use the concept of the *probability near mode* (PNM) of the distribution. It measures the amount of probability mass within some δ of the mode of the distribution. So, the robot's prediction of its success in going through the door would be the PNM with δ equal to half of the robot width minus the door width. We will use the following fluents to characterize sets of belief states over continuous variables. Although we later commit to a particular belief representation, the fluents are independent of the detailed underlying belief-state representation.

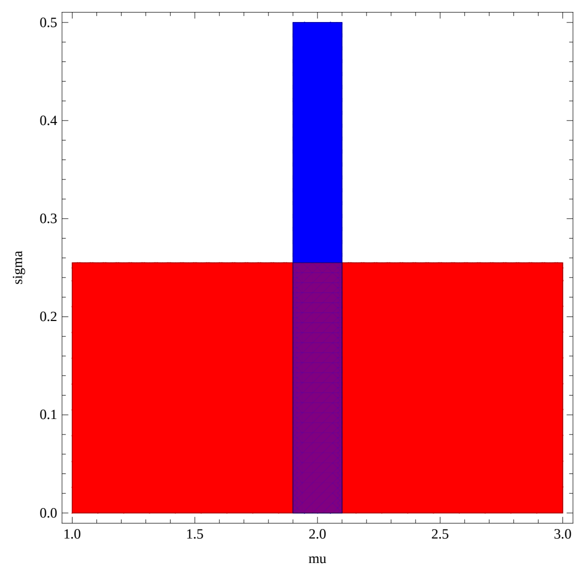
- $BV(X, \epsilon, \delta)$: *believe the value* of random variable X is within δ of its mode with probability at least $1 - \epsilon$. The corresponding test is
$$\tau_{BV}((X, \epsilon, \delta), b) := \Pr_b(|X - \overline{X}| < \delta) \geq 1 - \epsilon \ .$$
- $\mathit{ModeNear}(X, v, \delta)$: the mode of random variable X is within δ of value v. The corresponding test is
$$\tau_{\mathit{ModeNear}}((X, v, \delta), b) := |v - \overline{X}| < \delta \ .$$
- $B(X, v, \epsilon, \delta)$: the value of random variable X is within δ of value v with probability at least $1 - \epsilon$. The corresponding test is
$$\tau_B((X, v, \epsilon, \delta), b) := \Pr_b(|X - v| < \delta) \geq 1 - \epsilon \ .$$

Figure 17(a) illustrates these fluents in the case of belief distributions described in terms of a mean μ and standard deviation σ. The red region shows a set of belief states satisfying $BV(X, 0.05, 0.5)$; of course, since this fluent is independent of the mean of the distribution, the set extends infinitely along the μ axis. The blue region shows a set of belief states satisfying $\mathit{ModeNear}(X, 2, 0.1)$; in this case, the fluent is independent of the variance of the distribution so the set extends infinitely along the σ axis. The purple region is the intersection of the red and blue regions, and shows the belief states satisfying the conjunction $BV(X, 0.05, 0.5)$ & $\mathit{ModeNear}(X, 2, 0.1)$.

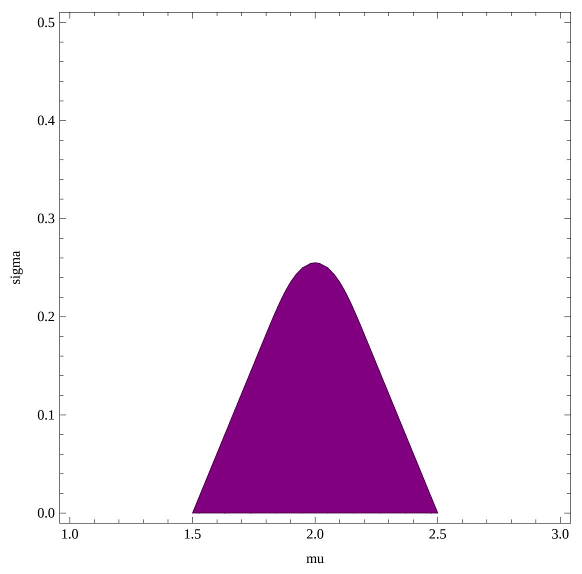
Figure 17(b) illustrates the belief set described by fluent $B(X, 2, 0.05, 0.5)$. This fluent trades off with more subtlety situations in which the mode is near and the variance high against situations in which the mode is farther away and the variance lower. However, it is more difficult to compute regression conditions for B than for BV and $\mathit{ModeNear}$, so we use BV and $\mathit{ModeNear}$ in the formalization of the robot domain.

5.3 Pre-images

There are two ways in which taking an action can effect these fluents: information can be gained through observation, or lost through actions with probabilistic effects; the mode can be changed by actions or observations. In general, actions will have both types of consequences, but we will illustrate them independently in the following.



(a) Regions of a Gaussian belief space captured by fluents: $BV(X, 0.05, 0.5)$ in red; $ModeNear(X, 2, 0.1)$ in blue; $BV(X, 0.05, 0.5)$ & $ModeNear(X, 2, 0.1)$ in purple.



(b) Regions of a Gaussian belief space captured by fluent $B(X, 2, 0.05, 0.5)$.

Figure 17: Fluents in continuous belief space

5.3.1 Degree of belief

For a planning goal of $BV(X, \epsilon, \delta)$, we need to know expressions for the regression of that condition under the a and o in our domain.

First, we determine such expressions for the case where the underlying belief distribution on state variable X is Gaussian, the dynamics of X are stationary (that is, that the value of X does not change as a result of the observation action), a is to make an observation, and the observation o is drawn from a Gaussian distribution with mean X and variance σ_o^2. The idea of characterizing a distribution in terms of PNM and computing its regression applies more generally, for example, to non-parametric distributions represented as sets of samples. In the non-parametric case, the pre-image function could not be computed analytically, but it might be possible to learn an approximate representation based on examples drawn from experience.

Here is an operator description for the effects of observation with variance σ_o^2 on the BV fluent, under the assumption that we always get an observation:[1]

OBSERVE(X):
effect: $BV(X, \epsilon, \delta)$
pre: $BV(X, obsRegress(\epsilon, \delta, \sigma_o), \delta)$
cost: 1

Somewhat surprisingly, the effect on the PNM of the belief state of making an observation is deterministic, even though the observation is stochastic. This is a special property of Gaussian obser-

[1]The observation noise in real robots is seldom actually independent and identically distributed (IID), and so this model of information gain based on successive observations is unrealistic.

vations, which is not true in general (as we already saw for discrete observations in the previous section). Additional preconditions could be added if there are conditions on getting an observation; the cost could be used to model situations in which getting an observation is probabilistic.

For a one-dimensional random variable $X \sim \mathcal{N}(\mu, \sigma^2)$, the probability within δ of the mode,

$$\textsc{pnm}(X, \delta) = \Phi\left(\frac{\delta}{\sigma}\right) - \Phi\left(-\frac{\delta}{\sigma}\right) = \operatorname{erf}\left(\frac{\delta}{\sqrt{2}\sigma}\right) \quad ,$$

where Φ is the Gaussian cumulative distribution function and erf is the *Gaussian error function*, which is a special function with a sigmoidal shape, satisfying

$$\Phi(x) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right) \quad .$$

Proposition 5.

$$\mathit{obsRegress}(\epsilon, \delta, \sigma_o) = 1 - \operatorname{erf}\left(\sqrt{\operatorname{erf}^{-1}(1-\epsilon)^2 - \frac{\delta^2}{2\sigma_o^2}}\right) \quad .$$

Figure 18(a) shows the regression of the condition $BV(X, 0.05, 0.5)$ (shown in red) under an observation with $\sigma_o = 0.4$: the regression is the union of the orange and red regions; we can see that a larger variance is allowable if the next action is an observation, because the observation will decrease the variance.

If the random quantity X is going to be changed by an amount u, then generally there is some loss of certainty about the value of X, characterized by transition noise σ_u, which may be dependent on the value of u. We can write an operator description for such an action:

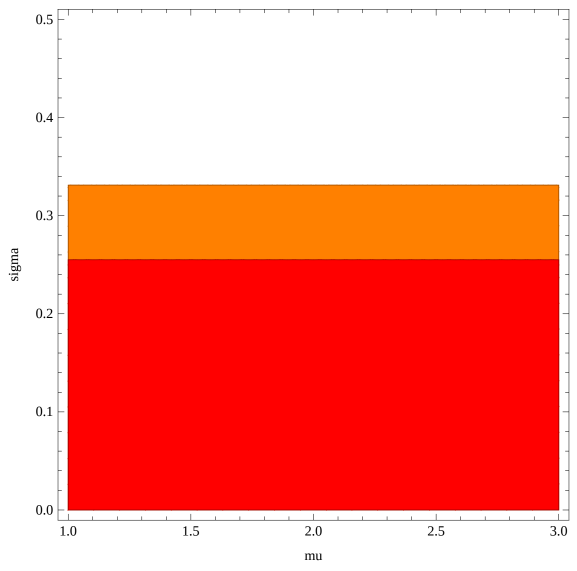
Change(X, u):
  effect: $BV(X, \epsilon, \delta)$
  pre: $BV(X, \mathit{changeRegress}(\epsilon, \delta, \sigma_u), \delta)$
  cost: 1

Although the results of changing X may be stochastic, the belief-state update is deterministic.

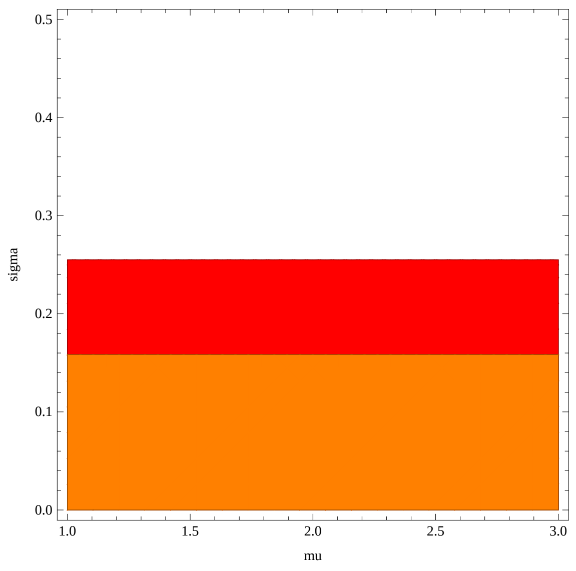
Proposition 6.

$$\mathit{changeRegress}(\epsilon, \delta, \sigma_Y) = 1 - \operatorname{erf}\left(\frac{\delta \operatorname{erf}^{-1}(1-\epsilon)}{\sqrt{\delta^2 - 2\sigma_Y^2 \operatorname{erf}^{-1}(1-\epsilon)^2}}\right) \quad .$$

If $\epsilon < 1 - \operatorname{erf}\left(\frac{\delta}{\sqrt{2}\sigma_Y}\right)$, then there is no degree of prior certainty that will guarantee that, after the action, the BV condition will be satisfied. Figure 18(b) shows the regression of the condition $BV(X, 0.05, 0.5)$ (shown in red) under a transition with $\sigma_Y = 0.2$: the regression is the orange region; we can see that only a smaller variance is allowable if the next action is a transition, because the motion will increase the variance.



(a) Red region: $BV(X, 0.05, 0.5)$; Union of red and orange regions: pre-image of the red region under the observation action, with $\sigma_o = 0.4$



(b) Union of red and orange regions: $BV(X, 0.05, 0.5)$; Orange region: pre-image of union of red and orange regions under the change action, with $\sigma_Y = 0.2$

Figure 18: BV fluents and pre-images under observation and change actions.

5.3.2 ModeNear

The regression of the *ModeNear* fluent under an action that moves the mode is easy to handle, but we will also have to consider its regression under observation actions, which is more difficult.

The regression of $ModeNear(v, \delta)$ under an action which changes the value by u is simply $ModeNear(v - u, \delta)$. Under the assumption of getting the most likely observation, the regression of any *ModeNear* fluent under an observation action is that same fluent, unchanged.

CHANGE(X, u):

effect: $ModeNear(X, v, \delta)$
pre: $ModeNear(X, v - u, \delta)$
cost: 1

However, in general, any other observation will move the mode of the distribution and may change the truth value of a *ModeNear* fluent. So, in fact, the probability that an observation will maintain this condition is related to the uncertainty in the distribution. The probability that an observation will violate a $ModeNear(v, \delta)$ condition depends on the uncertainty in the distribution at the time of the observation, as well as v and δ.

Proposition 7.

$$probModeMoved(\epsilon, \delta_b, v, \delta) = 2\Phi\left(\frac{\delta\sqrt{\sigma_r^2 + \sigma_o^2}}{2\sigma_r^2}\right) \quad ,$$

where

$$\sigma_r = \frac{\delta_b}{\sqrt{2}\,\mathrm{erf}^{-1}(1 - \epsilon)} \quad .$$

So, in the planning process, if an *Observe* action is being taken to achieve fluent $BV(\epsilon, \delta_b)$ and there is a $ModeNear(v, \delta)$ fluent in the goal at the same time, then the probability that the *ModeNear* fluent will be violated is $probModeMoved(\epsilon, \delta_b, v, \delta)$ and so we can rewrite the Observe operator as:

```
Observe($X$):
    effect: $ModeNear(X, v, \delta)$, $BV(X, \epsilon, \delta_b)$
    pre: $ModeNear(X, v, \delta)$, $BV(X, obsRegress(\epsilon, \delta_b, \sigma_o), \delta_b)$
    cost: $1 - \log probModeMoved(\epsilon, \delta_b, v, \delta)$
```

5.4 Example execution

To provide intuition about planning in belief space with a Gaussian belief on a single underlying continuous variable, we show two example planning and execution runs. We can think of this as a simple problem of navigation in one dimension, in which the robot can move by a continuous offset from its current position or make an observation of its current position with Gaussian noise. The goal is to have the mode of the distribution be near 5.0 and to have 0.95 of the probability mass within 0.4 of the mode.

Here are the operator descriptions for this domain. As discussed in [Kaelbling and Lozano-Pérez, 2012], the operator descriptions may, in general, depend on the goal γ in order to avoid choosing values that are in contradiction with the goal and on the current belief state b_{now} in order to choose values that may heuristically generate shorter plans.

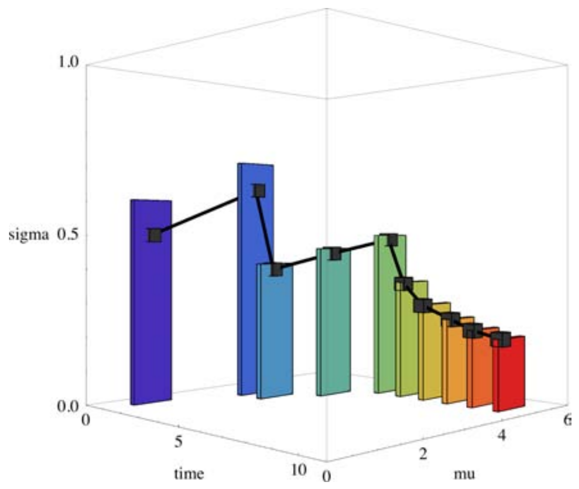
To limit the branching factor, the generator in the move operator considers actions that would move the mode by +1, -1, or by the total distance from the mode of the current belief distribution to the goal mode.

```
Move($t, u, \delta, b_{now}, \gamma$):
    effect: $ModeNear(t, \delta)$
    pre:
        choose: $u \in \{+1, -1, t - \overline{b_{now}}\}$
        $ModeNear(t - u, \delta)$
    prim: MovePrimitive($u$)
    cost: $|u|$
```

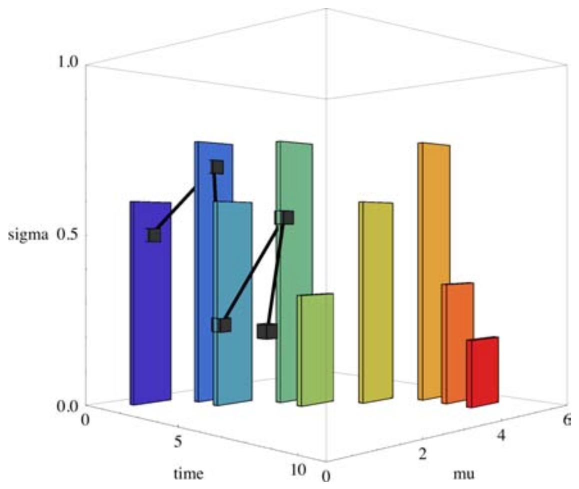
The standard deviation of action u is, $\sigma_u = \alpha|u|$, proportional to the magnitude of the control.

```
moveFluentRegress($fl, b$):
    if: $fl = BV(\epsilon, \delta)$
        return: $BV(changeRegress(\epsilon, \delta, \alpha|u|), \delta)$
    else:
        return: $fl$
```

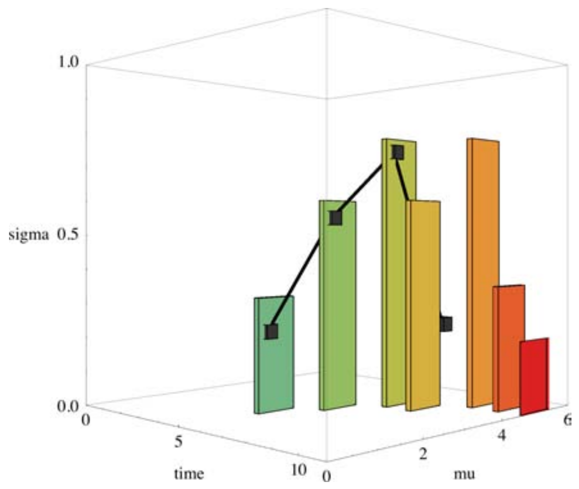
The observation action (look) has an additional precondition $BV(0.2, 1.0)$, which stipulates that, with probability 0.8, the location be known within 1.0; this simulates a requirement that the location be approximately known in order to successfully observe the object. So, a plan that does a large motion and then observes to verify it will not succeed: the large motion will increase the uncertainty so much that the observation action cannot be relied upon to see the object.



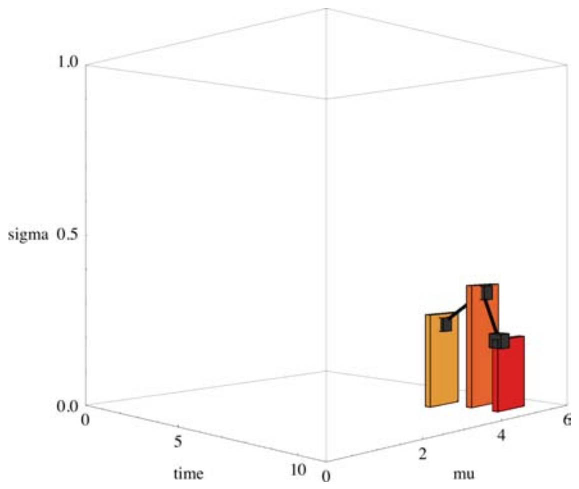
(a) Belief space plan, shown in colored bars and execution trajectory shown in black. The red bar is the goal in μ, σ space; the pre-images are shown going back through time.



(b) Belief space plan with higher transition error and lower observation error, in which the execution trajectory deviates from the plan.



(c) Plan made in response to deviation from plan in figure 19(b), in which execution trajectory deviates again.



(d) Final plan segment.

Figure 19: Example plans and execution trajectories in the one-dimensional continuous domain.

```
LOOK($\epsilon, \delta, b, \gamma$):
    effect: $BV(\epsilon, \delta_b)$, $ModeNear(v, \delta)$
    pre:
        $BV(obsRegress(\epsilon, \delta_b, \sigma_o), \delta_b)$
        $BV(0.2, 1.0)$
        $ModeNear(v, \delta)$
    prim: LOOKPRIMITIVE()
    cost: $1 - \log(probModeMoved(\epsilon, \delta_b, v, \delta))$
```

Figure 19(a) shows a sample planning and execution run, in which $\sigma_o = .5$ and $\alpha = 0.2$. This is relatively high observation error but low transition error. The plan is: MOVE(2), *Observe*, MOVE(1), MOVE(1), followed by 5 *Observe*s. The axes of the graph are the μ and σ of the belief and the time step of the process. The red bar at time 10 represents the goal $BV(0.05, 0.4)$ & $ModeNear(5.0, 0.5)$. The successive pre-images of the goal under the planned action sequence are shown in the bars moving backward in the time dimension. We can see that motion actions change the mean and also increase the variance. Because the observation error is high, it takes a long sequence of observations at the end to decrease the uncertainty sufficiently to achieve the BV goal. The black cubes denote the belief state; they are connected as they move through time.

Figures 19(b) through 19(d) shows a run with $\sigma_o = 0.25$ and $\alpha = 0.5$. In this case, fewer observations are necessary, but it elects to take only single-step actions in order to have opportunity to make observations that will keep the uncertainty down below the level at which observation actions are no longer applicable. Figure 20 shows the planning and replanning tree.

In figure 19(b), we see the initial plan, which is: MOVE(1), *Observe*, MOVE(1), *Observe*, MOVE(1), MOVE(1), *Observe*, *Observe*. On the fifth execution step, we can see that the belief state (a black cube near the middle of the figure) is not in the "envelope" (union of pre-images) of the plan, and so replanning is triggered. Figure 19(c) shows the new plan and belief space trajectory. On the fourth step of this plan, the execution again exits the envelope. A final plan, shown in figure 19(d), has a move and two observes, and is executed successfully.

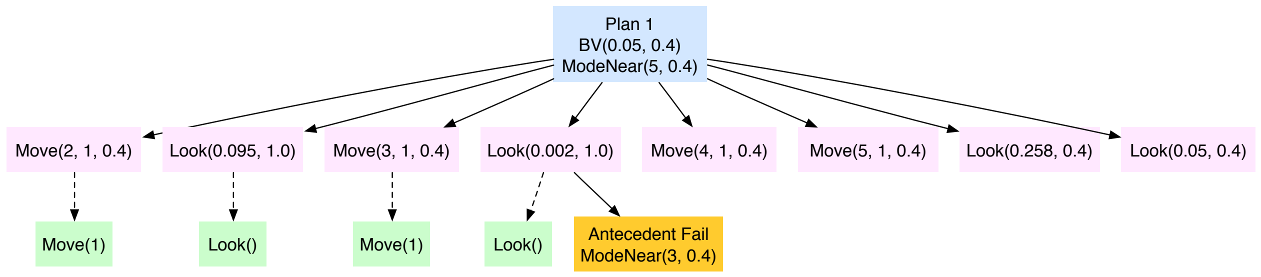
This example illustrates that we can effectively use discrete regression planning techniques in a logical representation belief space, and use execution monitoring and replanning to handle unexpected outcomes of those plans, eventually achieving a stated belief-space goal.

6 Pick-and-place domain

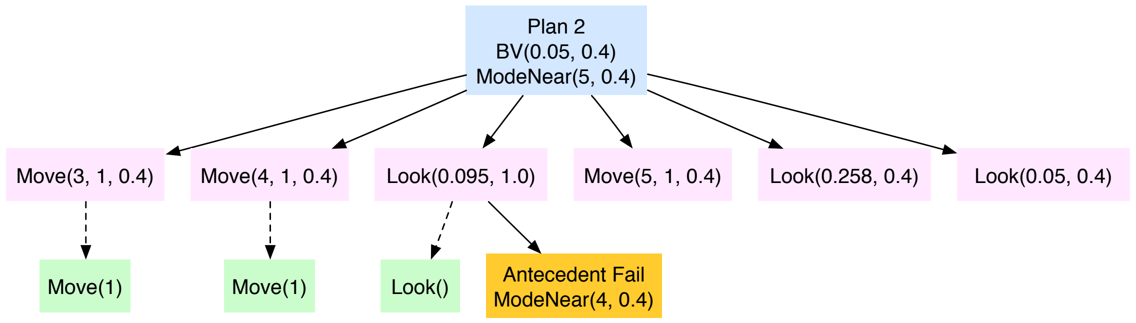
In this section, we show how the belief-space modeling and planning approach is used to control the robot manipulation domain described in section 2.

6.1 Primitives

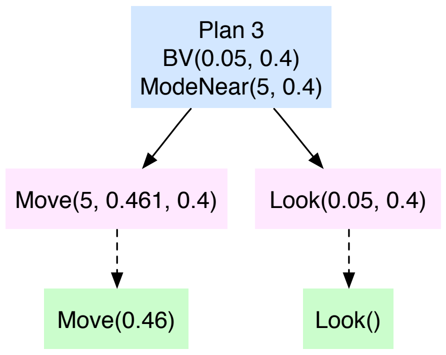
From the perspective of HPN, there are four primitive actions in this domain. In fact, most of the "primitives" actually involve calling a RRT-based robot motion planning algorithm [Kuffner and LaValle, 2000] to get a joint-space trajectory for the robot, smoothing the trajectory, and then executing it. The actual geometric planning is done in the maximum *a priori* probability (MAP) configuration of the objects and robot, extracted from the state estimator; this is known as the MAP *world*. The obstacles for the motion planner include the objects in the MAP world grown by a small margin to compensate for execution errors, and any unobserved cells in $\mathcal{S}_{obs}$.



(a) Plan 1



(b) Plan 2



(c) Plan 3

Figure 20: Planning and execution process for the one-dimensional continuous domain. The arguments to the MOVE action are the target location of the move, the distance being moved, and the δ representing how close the mode of the belief must be to the target location. The arguments to the LOOK action are the ϵ and δ such that the intended result of the action are to achieve $\text{PNM}(\delta) > 1 - \epsilon$.

- PICKPRIMITIVE(*obj*, *grasp*) causes the robot pick up object *obj* using grasp *grasp*, which is specified in terms of a pose for the object relative to the hand of the robot.
- PLACEPRIMITIVE(*region*) causes the robot to place the object it is currently holding at a pose that will ensure that the object is contained with the region of space *region*.
- MOVEROBOTPRIMITIVE(*conf*) moves the robot's base and arm to a particular configuration, *conf*, specified by a 3D pose for the base and a 4D pose for the hand (which is always oriented parallel to the floor, reducing the space from 6D to 4D).
- LOOKPRIMITIVE(*obj*) moves the head so that the centroid of object *obj* in the MAP world is in the center of the visual field of the robot. If the robot base is not in a configuration from which this is possible, it fails. It results in the perception system getting (noisy) observations of objects in the field of view.

6.2 Fluents

We use fluents to characterize beliefs about poses of objects, the configuration of the robot, whether the contents of geometric regions are known, and whether regions are clear. We begin by defining some concepts that will be used in the fluent definitions, then go on to define the fluents themselves.

6.2.1 Characterizations of uncertainty

In the previous development using PNM we were not committed to a particular distributional representation of uncertainty. For simplicity in the following, we will restrict ourselves to Gaussian uncertainty or to binary (known vs not-known) representations of uncertainty.

Recall from section 2.2 that the belief state of the mobile-manipulation robot consists of a joint Gaussian distribution on the 4-dimensional poses of all of the objects in the domain and the base pose of the robot. It also contains a point estimate of the hand pose of the robot and the gripper opening. These poses are all represented with respect to an arbitrary coordinate frame defined by the robot's starting pose; the variances may be large with respect to this initial coordinate frame and so the raw variances will never be a good measure of important information. Instead we are interested in how well we know the pose of one object relative to another; we will consider the distribution of the *difference* between the poses of two objects. Given a pair of scalar random variables with joint distribution $\mathcal{N}(\mu, \Sigma)$, the difference $X - Y$ is distributed as $\mathcal{N}(\mu_X - \mu_Y, \Sigma_{XX} + \Sigma_{YY} - 2\Sigma_{XY})$. So, for example, there may be significant uncertainty about both X and Y, evidenced in large values of Σ_{XX} and Σ_{YY}, but strong information about the correlation, evidenced in a large value of Σ_{XY}, resulting in low variance on the difference.

Pose difference distribution In the context of the overall state estimator, to compare poses P and Q, we extract four 4×4 sub-matrices of the overall covariance matrix: Σ_{PP}, Σ_{QQ}, Σ_{PQ} and Σ_{QP}. The notion of the difference between two poses is not as simple as subtraction of two reals. We can think of P as the pose of one object relative to a global frame and Q as the pose of another object relative to that same frame. Now, we are interested in the distribution of the pose of Q relative to P. Interpreting poses as rigid transformations, we are therefore interested in the distribution of $f(PQ) = Q^{-1}P$. Letting $PQ = [x_P, y_P, z_P, \theta_P, x_Q, y_Q, z_Q, \theta_Q]$, we have

$$f(PQ) = \begin{bmatrix} \cos(\theta_P)(X_Q - X_P) + \sin(\theta_P)(Y_Q - Y_P) \\ \cos(\theta_P)(Y_Q - Y_P) - \sin(\theta_P)(X_Q - X_P) \\ Z_Q - Z_P \\ |\theta_Q - \theta_P|_{\pm\pi} \end{bmatrix} ,$$

where $|\theta|_{\pm\pi}$ is the angle θ wrapped to be in the interval $[-\pi, +\pi]$.

We do not know a way to compute this distribution analytically. Instead, we use a sigma-point approximation [Julier and Uhlmann, 2004]. Define the α-sigma points of multivariate Gaussian distribution μ, Σ as follows:

```
SIGMAPOINTS($\mu, \Sigma, \alpha$):
    $\{\mu \pm \alpha\sqrt{\lambda_i} v_i \mid (\lambda_i, v_i) \in$ eigenvalues and normalized eigenvectors of $\Sigma\}$
```

These are deterministic samples of the joint distribution that effectively capture the covariance by extending α times the standard deviation along each of the axes of the covariance ellipsoid.

Define Σ_J to be the joint covariance matrix of P and Q,

$$\Sigma_J = \begin{bmatrix} \Sigma_{PP} & \Sigma_{PQ} \\ \Sigma_{QP} & \Sigma_{QQ} \end{bmatrix} .$$

Let

$$C = \textsc{sigmaPoints}(\overline{PQ}, \Sigma_J, 1) .$$

We can transform these sigma-points to be samples of $Q^{-1}P$, obtaining

$$D = \{f(c) \mid c \in C\} ,$$

then obtain an approximate distribution on $Q^{-1}P$ by fitting a Gaussian distribution to the points in D. We call this the *difference distribution* of poses P and Q, and write it μ_{P-Q}, Σ_{P-Q}.

Shadow The fluent test definitions make extensive use of the ϵ-*shadow* of object o_1 with respect to o_2 in a belief b. Define $\pi(o, b)$ to be the distribution in belief state b over the pose of object o. We use the pose-difference distribution, $\mu_{\pi(o_1,b)-\pi(o_2,b)}, \Sigma_{\pi(o_1,b)-\pi(o_2,b)}$ to construct a shadow, $S(o_1, o_2, \epsilon, b) = \textsc{makeShadow}(o_1, \mu_{\pi(o_1,b)-\pi(o_2,b)}, \Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \epsilon)$ as follows:

```
MAKESHADOW($o, \mu, \Sigma, \epsilon$):
    $n_{sd} = \sqrt{\chi^2_4(1-\epsilon)}$
    $\Pi$ = SIGMAPOINTS($\mu, \Sigma, n_{sd}$)
    for $j \in 1..numParts(o)$:
        $H_j$ = CONVEXHULL($\{vol(o_j, \pi) \mid \pi \in \Pi\}$)
    return: $\bigcup H_j$
```

We begin by finding n_{sd}, which is the number of standard deviations, with four degrees of freedom, one has to go out from the mean of a Gaussian distribution in order to contain $1 - \epsilon$ of the probability mass. This can be derived from the χ^2 distribution with 4 degrees of freedom. Next, we let Pi be the set of eight ϵ-sigma points of μ, Σ. Objects in our domain are defined to be the union of a set of convex parts; for each of these parts, we compute the convex hull of the volumes obtained by placing that part at each of the sigma points (which are poses). Finally, we return the union of these convex hulls.

The ϵ-shadow is constructed with the goal that, with probability $1 - \epsilon$, it contain the volume taken up by o if its pose is drawn from the given distribution. However, because we are using a polygonal approximation to what should really be a convolution of the shape of o with the $1 - \epsilon$ equi-probability ellipsoid of the distribution, the desired property is not guaranteed to hold.

Probability near mode for poses To express the degree of uncertainty in the distribution over the relative poses of two objects, we should develop a scalar PNM expression and regression procedures for four-dimensional distributions and apply them to relative poses. For expedience in our implementation, we define a vector-valued PNM_π which collects the PNM for the marginal distribution of each of the dimensions independently. Then, letting ϵ and δ be four-dimensional vectors of confidence and precision values, the expression

$$\text{PNM}_\pi(\Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \delta) \leq 1 - \epsilon$$

will be true if the PNM condition is satisfied componentwise.

6.2.2 Fluent definitions

In this section, we provide definitions of the fluents used to formalize the pick-and-place domain in terms of tests on belief states. Currently, all of the conditions reduce to requirements on the pose of the object or robot. The approach is readily extended to other types of conditions, such as those involving contacts, forces, and containment.

- $BVRelPose(o_1, o_2, \epsilon, \delta)$: true if the distribution on the difference between the poses of objects o_1 and o_2 satisfies the following requirement, where ϵ and δ are both vectors of four values:

$$\tau_{BVRelPose}((o_1, o_2, \epsilon, \delta), b) := \text{PNM}_\pi(\Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \delta) \leq 1 - \epsilon \ \ .$$

- $PoseModeNear(o, \pi, \delta)$: true if the mode of the distribution of the pose of o is within δ of pose π, where δ is a vector of four values.

$$\tau_{PoseModeNear}((o, \pi, \delta), b) := |\overline{\pi(o, b)} - \pi| \leq \delta \ \ ,$$

  where the difference is a 4-dimensional vector of componentwise absolute differences (with the angular difference appropriately wrapped), and $\leq$ holds for the vector if it holds for all four components individually.

- $ConfModeNear(c, \delta)$: true if the mode of the distribution of configuration of the robot is within δ of the configuration c. In this case, it tests to see that both the 3D Cartesian pose of the base and the 4D Cartesian pose of the hand are within δ of those specified in c (in the same manner as for *PoseModeNear*) and that the actual gripper opening is within a global δ_{grip} of the gripper opening in c.

- $BIn(o, r, \epsilon)$: true if the probability that object o is entirely contained within a geometric region r is greater than $1 - \epsilon$. This is determined by testing to see whether the ϵ shadow of o with respect to the region is contained in the region:

$$\tau_{BIn}((o, r, \epsilon), b) := S(o, r, \epsilon, b) \subseteq r$$

- *BContents*(r, x, ϵ): true if the contents of region r are known with reasonably high certainty.

  This is fluent is tested in two different ways, depending on whether we are using an explicit representation of the space that has been observed, $\mathcal{S}_{obs}$. If we are not representing $\mathcal{S}_{obs}$, then we assume we are aware of the existence of all objects in the domain, and require that all objects whose ϵ shadows overlap r, except those in the list of exceptions x, have high-certainty pose estimates with respect to r. That is:

  $$\tau_{BContents}((r, x, \epsilon), b) := \forall o \in (\{o \mid S(o, r, \epsilon, b) \cap r \neq \emptyset\} - x).\ \tau_{BVRelPose}((o, r, \epsilon_{bc}, \delta_{bc}), b) \ ,$$

  where ϵ_{bc} and δ_{bc} are four-dimensional vectors of fixed confidence parameters. If we have an explicit representation of $\mathcal{S}_{obs}$, then we simply test to see whether the region has been observed.

  $$\tau_{BContents}((r, x, \epsilon), b) := r \subseteq \mathcal{S}_{obs} \ .$$

- *BClearX*$((r, x, \epsilon), b)$: true if region r is known with confidence $1 - \epsilon$ to be clear of obstacles, except for those in set x:

  $$\tau_{BClearX}((r, x, \epsilon), b) := \tau_{BContents}((r, x, \epsilon), b)\ \&\ \forall o \notin x.\ S(o, r, \epsilon, b) \cap r = \emptyset \ .$$

  In fact, this is not strictly correct. If there were a single object, then if its ϵ shadow does not overlap the region, then with probability at least $1 - \epsilon$ the region is clear. With multiple possible overlapping objects, the confidence parameter should be made smaller using something like a Bonferroni correction.

- *BHolding*(o, g, ϵ) : true if object o is currently being held by the robot with grasp g with probability ϵ.

  $$\tau_{BHolding}((o, g, \epsilon), b) := (o = \mathit{heldObject}\ \&\ |g - \mathit{heldObjectGrasp}| < \delta_g) \ ,$$

  where the global δ_g is a 4-dimensional vector of grasp pose tolerances. This definition reflects the current implementation which does not maintain an estimate of the uncertainty of the grasp pose.

The entailment and contradiction methods necessary to support planning in this infinite domain are described in appendix A.2.

6.3 Operator descriptions

In this section we provide the operator descriptions that specify the dynamics of each operation in the domain. The operations are: picking up an object, placing an object, moving the robot base, coming to know the contents of a region, clearing a region, and looking at an object. These operator descriptions characterize one useful set of trajectories through belief space. They cannot possibly be complete, in the sense of characterizing all possible trajectories.

It is important to remember that each operator description is, in fact, a *schema*: that is, it stands for an infinite class of operator descriptions, one for each binding of the parameters and the **choose** variables. When picking values of confidence (ϵ) and precision (δ) parameters, there is a continuous space of possibilities. We can set them, for any given operator, so that operator is likely to succeed and hence have low negated log probability cost. However, that will make the rest of the plan more expensive. In the following, we use some fixed values that we found to be satisfactory; in future, one could imagine learning good values and/or searching over more choices.

In all of the following operator descriptions, the belief state at the time the plan is being constructed is b_{now} and the current subgoal during backchaining is γ; b_{now} is used in the generators to give heuristic guidance and γ is used in the generators to provide constraints that avoid generating choices that would cause contradictions with the current goal.

6.3.1 Pick

Following is the description of the Pick operation; it is slightly simplified here, omitting some considerations that allow regrasping. It takes two arguments: o is an object to be picked and g is the grasp with which it is to be picked. If the grasp variable is unbound, it will be bound by a generator. The result of this operation is that the robot believes with high probability that it is holding object o with grasp g.

```
$\textsc{Pick}(o, g, b_{now}, \gamma)$:
    **effect:** $\mathit{BHolding}(o, g, \epsilon)$
    **pre:** $o \neq \mathit{None}$    **[0]**
        $\mathit{BVRelPose}(o, \textsc{robot}, \epsilon_{pickPlan}, \delta_{pickPlan})$    **[1]**
        **choose:** $\pi \in \{\pi(o, b_{now}), \textsc{generateParking}(o, g, b_{now}, \gamma)\}$
        **choose:** $\mathit{motion} \in \textsc{generatePickPath}(o, g, l, b_{now}, \gamma)$
        $\mathit{PoseModeNear}(o, \pi, \delta_{pickPlan})$    **[2]**
        $\mathit{BClearX}(\mathit{sweptVolume}(\mathit{motion}), \{o\}, \epsilon_{clear})$    **[2]**
        $\mathit{BHolding}(\mathit{None}, \mathit{None}, \epsilon_{holding})$    **[3]**
        $\mathit{ConfModeNear}(\mathit{pregraspConfig}(\mathit{motion}), \delta_{grasp})$    **[3]**
        $\mathit{BVRelPose}(\textsc{robot}, o, \epsilon, \delta_{grasp})$    **[4]**
    **sideEffects:**
        $\mathit{ConfModeNear}(C, D) = \mathit{None}$    **[0, 1]**
    **prim:** $\textsc{PickPrimitive}(o, g)$
```

The main section of the operator description is a list of preconditions; each precondition is optionally followed by a number indicating at what abstraction value of hierarchical planning that precondition is being considered. At the most abstract value (0) the only requirement is that o not be *None*; that just means that this operation cannot be used to empty the hand.

At abstraction value 1, the precondition is that the relative pose of the object and the robot be known with relatively weak certainty. Only when this condition is achieved can we plan in detail for how to pick the object.

At the next level of abstraction, the operator considers two poses, π, from which the object might be picked up: the first is the pose of the object in the current belief state, b_{now} (note that this is only evaluated after the value-1 belief condition has been achieved), and the second is a "parking" place. Picking an object up from parking allows us the option of constructing a plan that is non-monotonic in the sense that it moves the object (at least) twice [Stilman and Kuffner, 2006]: first, from its current pose to a parking location, and now, from the parking location to the hand. The parking location is computed by a generator procedure, which considers the the object and grasp, as well as the current goal, γ.

For each location, we call another generator, which determines a final pre-grasp configuration for the robot and a path through configuration space from a home configuration to the pre-grasp configuration. Paths are always planned to and from the home configuration. The preconditions ensure that the path is traversible, and because we always guarantee this condition, the robot is guaranteed never to block itself in. However, when the primitive is actually executed, it calls a planner again and attempts to move as directly as possible to the pre-grasp pose, without necessarily going via the home configuration. The path generator is very similar to the generators used in deterministic HPN. It treats objects in their poses in the MAP environment as well as their ϵ shadows as potentially movable obstacles, and treats the complement of $\mathcal{S}_{obs}$, if it is represented, as a permanent obstacle. It prefers to generate paths that avoid movable obstacles, but will go through them if necessary (necessitating that they be removed, either through sensing, which reduces the sizes of shadows or by physically moving objects out of the way).

Given a generated motion, there are several preconditions with different abstraction values. There are many different ways to organize the precondition hierarchy; this is one version that works effectively.

With abstraction value 2, we require that the mode of the pose distribution for object o be near l; this condition was true when the motion was constructed, and we need to ensure that if, during execution of this operation, observations are made of o that cause its estimated pose to change considerably, this part of the plan fails and is recomputed. Also at this level we require that the swept volume of the path be believed to be clear, with the exception of object o. If the volume of space had not been carefully observed before, this precondition will require coming to know the contents of the region. If it was impossible to find a path that did not collide with objects in their current poses, then achieving this precondition will require moving objects out of the way.

With abstraction value 3, we require that the robot not be holding anything and that the base pose be near the required pre-grasp configuration of the motion. Finally, at value 4, we require that the pose of the object relative to the robot be known to sufficiently high accuracy as to achieve the required confidence about the grasp.

In addition to preconditions and results, we also specify side effects, characterizing *possible* effects of abstract versions of the operator. Thus, with abstraction values 0 and 1, we assert that any fluent that matches the pattern $\mathit{ConfModeNear}(C, D)$ will be affected unpredictably by this operation: we are not yet sure where the robot will be located. This side effect is not necessary for values 2 or higher, because at that level we have an explicit pre-condition stating what the robot's configuration will be.

The cost should be a function of the failure probability, which depends in a complex way on ϵ, δ_{grasp}, ϵ_{clear}, $\epsilon_{holding}$, $\epsilon_{pickPlan}$, and $\delta_{pickPlan}$. We have not yet determined the appropriate relationship and simply use a cost of 1; it will be an important topic of future work to learn to predict cost as a function of the situation. The ϵ in the result is controlled by the ϵ in the required knowledge of the relative position of robot and object before the pick primitive is executed. We are currently not modeling the probability that the pick operation will fail entirely.

6.3.2 Place

The place operation is similar to pick. The result is that the mode of the pose of o is near a specified pose. The target pose is defined relative to a physical object, and therefore may move as the object is observed and the belief state is updated.

```
PLACE($o, \pi, region, \epsilon, b_{now}, \gamma$):
    effect:
        $PoseModeNear(o, \pi, \sigma_{objPlace})$
    pre:
        $BVRelPose(o, \text{ROBOT}, \epsilon_{pickPlan}, \delta_{pickPlan})$    [1]
        choose: $motion \in \text{GENERATEPLACEPATH}(o, region, b_{now}, \gamma)$
        $BClearX(sweptVolume(motion), \{o\}, \epsilon_{clear})$    [2]
        $BHolding(o, grasp(motion), \epsilon_{holding})$    [3]
        $ConfModeNear(preplaceConfig(motion), \delta_{grasp})$    [4]
        $BVRelPose(\text{ROBOT}, region, \epsilon, \delta_{grasp})$    [4]
    sideEffects:
        $ConfModeNear(C, D) = None$    [0, 1, 2, 3]
        $BVRelPose(o, O_2, E, D) = None$    [0, 1, 2, 3, 4]
    prim: PLACEPRIMITIVE($\pi$)
```

The precondition with abstraction value 1 is that the pose of the object with respect to the robot be known, but with low precision; having this information allows the generator to select a grasp for placing the object that is also feasible for picking up the object, if possible. An alternative formulation in which we do not try to coordinate the grasp for the pick and the place is possible; it may result in the robot needing to re-grasp the object when it is time to perform the place, however.

Based on knowledge of the object's current pose, we generate one or more candidate place motions: each motion contains a grasp for the object, a pose for the object within the region, a configuration that the robot should be in before calling the PLACEPRIMITIVE, and a path through configuration space from the home configuration to the pre-place configuration. The place-path generator has similar constraints and preferences to the pick-path generator.

The remaining conditions ensure that the swept volume for the motion is clear, that the robot is holding the object in the correct grasp, that the robot is in the pre-place configuration, and that the robot's pose with respect to the region is known accurately.

As with the PICK operator, the abstract versions of this operator have a non-deterministic side-effect, not knowing where the robot will be at the end; in addition, we are unsure how accurately we will know the relative pose between o and the other objects in the environment. This side effect is quite strong, in the sense that it invalidates the knowledge of all relationships between o and other objects; it should probably only invalidate those that require a tolerance less than the sum of the uncertainty between o and the robot and between O_2 and the robot.

6.3.3 Move robot

This operation moves the robot to a desired configuration. It is impossible to guarantee a tolerance δ tighter than the tolerance of the underlying controller.

```
MOVEROBOT($c, \delta, b_{now}, \gamma$):
    effect: $ConfModeNear(c, \delta)$
    pre: $\delta \geq \delta_{controller}$    [0]
        choose: $motion \in \text{GENERATEROBOTMOTION}(c, b_{now}, \gamma)$
        $BClearX(sweptVolume(motion), \{\ \}, \epsilon_{clear})$    [1]
    prim: MOVEROBOTPRIMITIVE($c$)
```

When the robot moves, our information about the relative poses of the robot and other objects changes.

```
MoveRegress($fl, b_{now}$):
    if: $fl = BVRelPose(o_1, o_2, \epsilon, \delta)$ & $(o_1 = \textsc{robot} \mid\mid o_2 = \textsc{robot})$
        $\epsilon_r = changeRegress(\epsilon, \delta, \sigma_u)$
        if $\epsilon_r$:
            return: $BVRelPose(o_1, o_2, \epsilon_r, \delta)$
        else:
            return: False
    else:
        return: $fl$
```

In this case, we change *BVRelPose* fluents. If one of the objects is the robot, then moving the robot will weaken our knowledge about the relative pose of the robot and the other object. So, the *changeRegress* function defined in section 5.3.1 is applied componentwise to 4D vectors of values ϵ, δ, and σ_u, yielding a 4D vector ϵ_r which represents a more stringent condition that is the pre-image of the original condition. If ϵ_r is not defined, then we return False, indicating that the regression of the condition under this operation is false, and thus not a legal move.

6.3.4 Look at an object

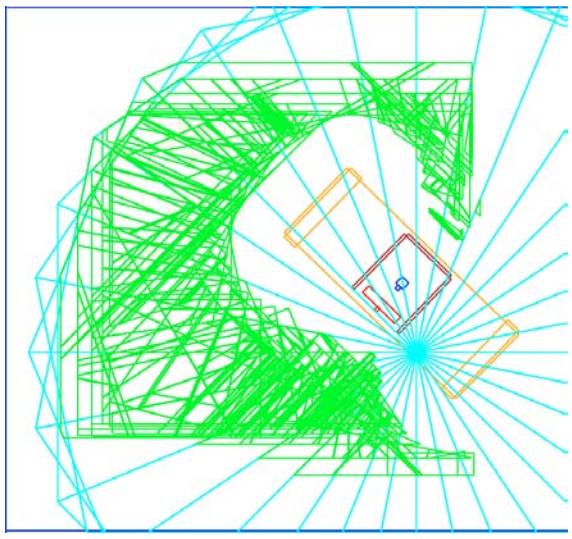
The operation of looking at an object can be used to achieve a *BVRelPose* condition. For simplicity we will elide some low-level details: some of the "objects" in our universe are regions in space, which have fixed poses relative to other physical objects in the universe whose poses are not known with certainty. When we wish to improve a relative pose estimate involving a non-physical object, we will, in fact, look at the physical object with respect to which it is defined. For the purposes of the definitions written below, we will assume that all relevant objects are physical.

To come to know the pose of one object relative to another, there are essentially two methods: to observe them both at the same time, or to observe them sequentially without inducing too much error in the relationship between the robot's poses at those observations. Here, we take the simple, but weak approach of reducing the requirement that we know the pose of o_1 with respect to o_2 within δ to two requirements: that we know pose of each of o_1 and o_2 with respect to the robot within $\delta/\sqrt{2}$.

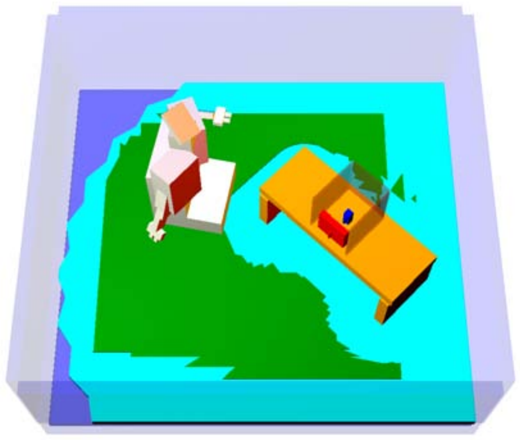
These particular conditions are not necessary, however: it could be that it is more effective, for some reason, to spread the uncertainty differently between the two objects.

```
LookObjSplit($o_1, o_2, \epsilon, \delta, b_{now}, \gamma$):
    effect: $BVRelPose(o_1, o_2, \epsilon, \delta)$
    pre: $\epsilon > 0$    [0]
        $o_1 \neq \textsc{robot}$ & $o_2 \neq \textsc{robot}$    [1]
        $BVRelPose(o_1, \textsc{robot}, \epsilon, \delta/\sqrt{2})$    [1]
        $BVRelPose(o_2, \textsc{robot}, \epsilon, \delta/\sqrt{2})$    [1]
```

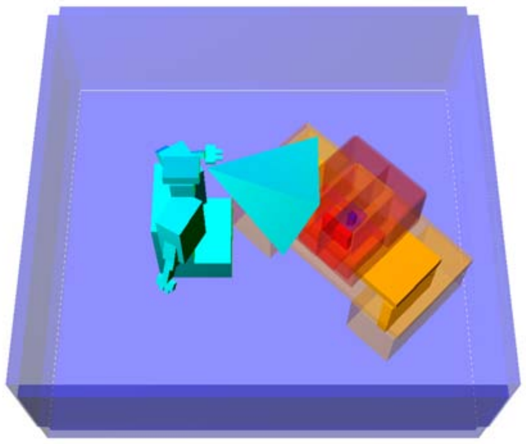
The following operator description characterizes conditions for coming to know the pose of an object with respect to the robot.



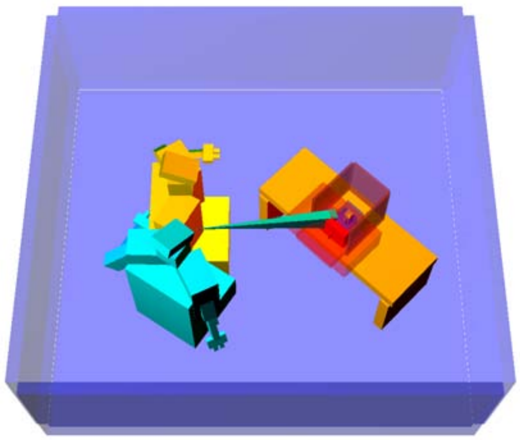
(a) Two-dimensional view of candidate look free space.



(b) Three-dimensional view.



(c) View cone from current location for the cupboard.



(d) View cone from different location (in cyan) for the blue cup; the cone intersect the red obstacle, which would need to be moved.

Figure 21: Candidate view locations used in generating look motions.

```
$\textsc{LookObj}(o, \textsc{robot}, \epsilon, \delta, b_{now}, \gamma)$:
    **effect:** $BVRelPose(o, \textsc{robot}, \epsilon, \delta)$
    **pre:** $\epsilon > 0$    **[0]**
    **choose:** $(motion, viewCone) \in \textsc{generateLookMotion}(o, b_{now}, \gamma)$
    **pre:** $BClearX(viewCone, [o], \epsilon_{huge})$    **[1]**
        $ConfModeNear(viewConf(motion), \delta_{\textsc{look}})$    **[1]**
        $BVRelPose(o, \textsc{robot}, obsRegress(\epsilon, \delta, \sigma_o), \delta)$    **[2]**
    **prim:** $\textsc{LookAtObjPrimitive}(o)$
```

The generator for look motions finds a configuration for the robot base, arm, and head so that it can see the center of the object. The first step is to characterize (x, y, θ) configurations of the base so that the target point is in the field of view. For a given value of θ, the (x, y) positions define an isosceles triangle whose apex is the target point; a large number of these triangles are shown in figure 21(a). Then, each of these triangles is cropped using the configuration space obstacles for the base relative to the objects in the environment. The resulting free space is a set of overlapping polygons in figure 21(a). These polygons represent configurations for the base such that the target point is in the field of view of the sensor but it does not take into account the presence of other objects. For example, when the robot is in the configuration shown in figure 21(b) it cannot actually see the target location. The generator samples locations in the valid free space regions and does a detailed test for visibility to find candidate view-point configurations.

The generator then constructs a path from the home configuration to the view-point configuration, and also constructs a *view cone*, which is really a frustum extending from the camera in the direction of the object to be viewed, and truncated a little short of reaching the object along the view direction. When selecting the view-point configuration and the path, the following considerations are taken into account:

- The swept volume of the robot along the path **must not** collide with any permanent object or with any object that is in a place required by subgoal γ.
- The view cone **must not** collide with any part of the robot (including the arms), with any permanent object or with any object that is in a place required by subgoal γ.
- The swept volume of the robot along the path is preferred not to collide with the ϵ shadows of objects in the belief state b_{now}.
- It is preferred not to move the robot base.

Avoiding shadows in the swept volume is important, if possible, because the robot will be required to verify that the volume is clear before it moves through it, and it can get permanently stuck if it tries to move into the shadow of an object in order to view that object to reduce its shadow. There are several other ways we could incorporate the current belief state into the results of the generator, including attempting to generate a view cone that covers more of a shadow of the target object, rather than simply its most likely pose. This is, again, a trade-off between the difficulty of planning and executing the view (the larger the region, the more likely it is to be occluded) against the likelihood that the object will be seen.

The *obsRegress* procedure here is as defined in section 5.3.1, but applied componentwise for all four components. As in that section, this operator should have an additional penalty derived from the probability that it will violate a *ModeNear* condition in γ. However, the current implementation does not include that.

6.3.5 Exploring a region

We approach the problem of making *BContents* fluents true differently depending on whether we know the universe of objects and have pose estimates for them, or we have an explicit representation of $\mathcal{S}_{obs}$.

When the universe of objects is known, the operator to make a *BContents* fluent true is *definitional*, in the sense that it does not correspond to any primitive action; instead, it is a way to reduce the *BContents* condition to a conjunction of *BVRelPose* conditions. The idea is that for every object that has a possibility greater than $1 - \epsilon$ of overlapping r, it is necessary to know the pose of that object with reasonably high accuracy with respect to r. In fact, this could be relaxed, with a special ϵ and δ for each object which would suffice to keep its shadow from overlapping the region in question. LOOK operations will be able to make these *BVRelPose* conditions true.

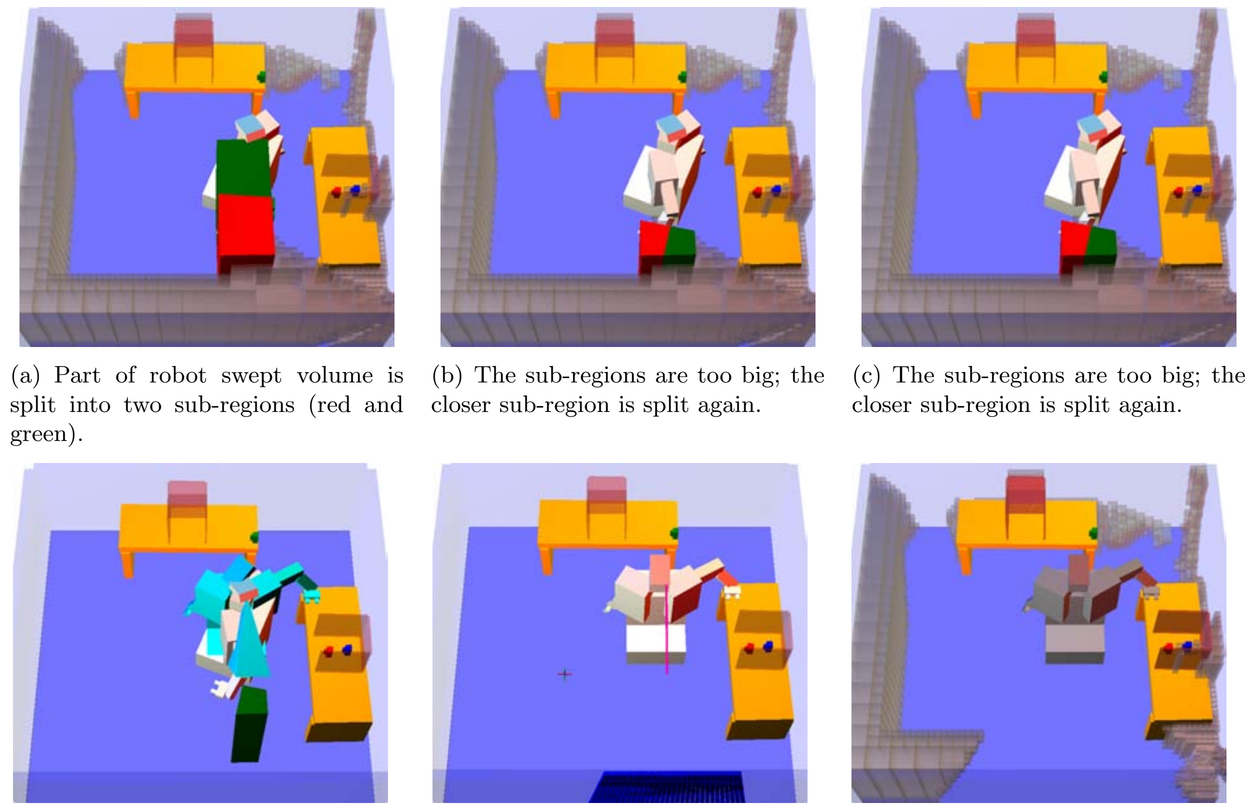
```
BCONTENTS1($r, x, \epsilon, b_{now}, \gamma$):
    effect: $BContents(r, x, \epsilon)$
    pre: $\epsilon > 0$    [0]
        let: $occluders = \{o \mid S(o, r, \epsilon, b_{now}) \cap r \neq \emptyset\} \setminus x$
        $\forall o \in occluders.\ BVRelPose(o, r, \epsilon, \delta_{bc})$    [1]
```

In the case that we are representing $\mathcal{S}_{obs}$ explicitly, we depend on a recursive decomposition of the region, illustrated in figure 22. If the region can be viewed in one look operation, then we generate a configuration for the robot and associated view cone for viewing the region and require the view cone not to be known to be occluded and require the robot configuration to be near the generated one; if the region is too big for a single view, then we split it into subregions, driven partly by the desire to aggregate space that has already been viewed into the same subregion and space that has not been viewed into different subregions. For each of the subregions, we assert that the contents must be known.

```
BCONTENTS2($r, x, \epsilon, b_{now}, \gamma$):
    effect: $BContents(r, x, \epsilon)$
    pre: $\epsilon > 0$    [0]
    if $smallEnoughToView(r)$:
        choose: $(motion, viewCone) \in$ GENERATELOOKMOTION($r, b_{now}, \gamma$)
        pre: $BClearX(viewCone, [o], \epsilon_{huge})$    [1]
            $ConfModeNear(viewConf(motion), \delta_{\text{LOOK}})$    [2]
        prim: LOOKATOBJPRIMITIVE($r$)
    else:
        let: $subregions = split(r, b_{now})$
        pre: for $s \in subregions$:
            $BContents(s, x, \epsilon)$    [1]
```

6.3.6 Clearing a region

Coming to believe a region is clear is handled in a similar way. It currently makes use of a special region in space, called the *warehouse*, which is known to be "out of the way," in the sense that there are no goal object placements in that region. Having such a region is not crucial, and we could rely on a placement generator to pick placements in the world instead.


(a) Part of robot swept volume is split into two sub-regions (red and green).

(b) The sub-regions are too big; the closer sub-region is split again.

(c) The sub-regions are too big; the closer sub-region is split again.

(d) A look pose (cyan) for the robot to stand so as to look at the target region (green) is generated.

(e) The sensing operation is carried out and nothing is observed.

(f) The global oct-tree is updated to reflect the new information.

Figure 22: Determining the contents of a swept volume.

```
BClearX($r, x, \epsilon, b_{now}, \gamma$):
    effect: $BClearX(r, x, \epsilon)$
    pre: $\epsilon > 0$    [0]
        $BContents(r, x, \epsilon)$    [1]
    let: $occluders = \{o \mid S(o, r, \epsilon, b_{now}) \cap r \neq \emptyset\} \setminus x$
    pre: $\forall o \in occluders.\ BIn(o, warehouse, \epsilon, \delta_{bc})$    [2]
        $BClearX(r, x \cup occluders, \epsilon)$    [2]
```

This operation is definitional and has no corresponding primitive. With abstraction value 1, it requires that the contents of r be known fairly accurately. At the next level, it generates a conjunction of requirements that all movable objects known to overlap the region that are not in the list of exceptions be in the warehouse. (In fact, this condition is too strong; it is enough for them to not overlap r.) Finally, we include a condition that there be nothing else in region r except for the exceptions and the list of *occluders* that we just found: this protects the region against having other objects put into it by operations found later in the planning process.

6.3.7 Putting an object in a region

We define an operator with no primitive that reduces a $\mathit{BIn}(o, r, \delta)$ condition to the conjunction of $\mathit{PoseModeNear}(o, \pi, \delta_{rp})$ and $\mathit{BVRelPose}(o, r, \epsilon_{rp}, \delta_{rp})$. There are generally many combinations of choices of π, ϵ_{rp}, and δ_{rp} that make this reduction valid. For simplicity in the current implementation, we fix ϵ_{rp} and δ_{rp}, and will seek an appropriate placement π.

We use $\textsc{epsDeltaShadow}(o, \epsilon, \epsilon_{rp}, \delta_{rp})$ to specify a shadow region sh, such that if the mode of the pose distribution for o is at the centroid of sh and the variance of the pose distribution for o, Σ, is such that $\textsc{pnm}_{\pi}(\Sigma, \delta_{rp}) > 1 - \epsilon_{rp}$, then, with probability $1 - \epsilon$, o is contained in sh. Then, because the *PoseModeNear* and *BVRelPose* conditions would guarantee that the object is in this shadow with high probability, it would be sufficient to find a placement such that the shadow is contained in r.

An approximation to the exact shadow computation can be made as follows. For each dimension ϕ of the pose, individually, we find a standard deviation

$$\sigma_{\phi} = \frac{\delta_{rp}[\phi]}{\sqrt{2}\,\mathrm{erf}^{-1}(1 - \epsilon_{rp}[\phi])} \quad .$$

We then use these values to make a diagonal covariance matrix Σ which satisfies the $\textsc{pnm}$ condition above. Then, we define $\textsc{epsDeltaShadow}(o, \epsilon, \epsilon_{rp}, \delta_{rp})$ to be $\textsc{makeShadow}(o, \mathbf{0}, \Sigma, \epsilon)$.

Now we have a shadow region for the object so that the $\textsc{pnm}$ conditions on the object's distribution will guarantee that it is within the shadow; now we only need to find a placement for the shadow inside the region, which we do by calling $\textsc{generatePlacePath}$ to find a placement for that shadow in r.

```
PutIn($o, r, \epsilon, b_{now}, \gamma$):
    effect: $\mathit{BIn}(o, r, \epsilon)$
    let: $sh = \textsc{epsDeltaShadow}(o, \epsilon, \epsilon_{rp}, \delta_{rp})$
    choose: $\mathit{motion} \in \textsc{generatePlacePath}(sh, r, b_{now}, \gamma)$
    pre: $\mathit{PoseModeNear}(o, \mathit{targetPose}(\mathit{motion}), \delta_{rp})$    $[\mathbf{0}]$
         $\mathit{BVRelPose}(o, r, \epsilon_{rp}, \delta_{rp})$    $[\mathbf{0}]$
```

There are many trade-offs here: for example, if we know there is lots of room in the region, we might want to do the placement with very low precision and select a pose near the middle of the region; or, if we think space will be tight, we might want to place very precisely. If we make δ_{rp} too small, we have trouble doing the place. If we make ϵ_{rp} too small, we will have to look many times to verify that the object is placed sufficiently precisely. So, we have tuned the values of δ_{rp} and ϵ_{rp} so that they are plausibly achievable, and work with the shadow size that results.

6.3.8 Emptying the hand

We make immediate use of the operation of putting an object in a region when we wish to empty the robot's hand of something. We reduce the condition of not holding anything to the condition of having the currently held object be contained in the *warehouse* region. Of course, this condition is sufficient, but not necessary, and one could imagine relaxing it.

PUTDOWN($\epsilon, b_{now}, \gamma$):
  effect: $BHolding(None, None, \epsilon)$
  pre: $BIn(currentlyHeldObject(b_{now}), warehouse, \epsilon)$ **[0]**

7 Example executions

The companion material to this paper includes movies illustrating sequences of robot actions, calls to motion-planning generators, and belief states for each of the following examples. The simulated examples all include uncertainty in motions and perception.

7.1 Detailed example of pick and place

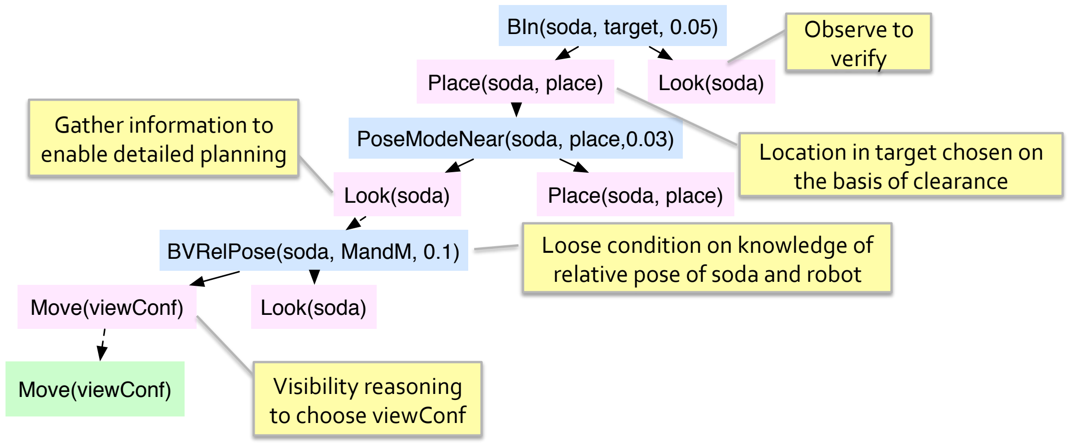
Figure 24 shows a slightly simplified HPN planning and execution tree for a simple example involving a blue box (referred to as *soda*) and a red can (referred to as *soup*). At the top of each sub-figure is the relevant part of the tree: planning goals are blue, operator instances are pink, primitive actions are green, and indications of replanning steps are yellow and orange. In the plan, the robot is referred to as $MandM$[2]. The bottom of the figure shows a sequence (left to right) of snapshots of the belief state and a corresponding image of the actual execution on the PR2 robot.

The robot's goal is to place the soda box in a specified (target) region on the left side of the table. The red can (soup) blocks access to valid grasps for the soda box (recall that we restrict the choice of grasps to those in which the hand is parallel to the support plane). In this example, we are not explicitly modeling $\mathcal{S}_{obs}$, and all the objects in the universe are known to the robot, but with high pose uncertainty.

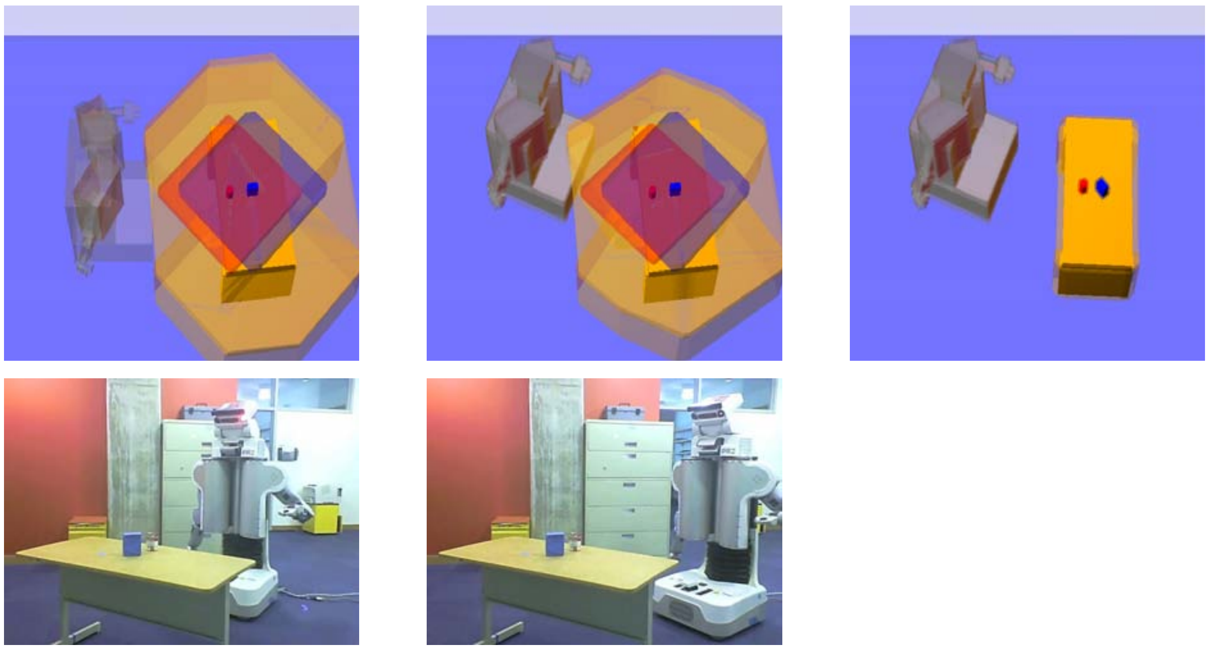
The initial plan for placing the soda in the target region is shown in figure 23(a). At the highest level of abstraction, it is to PLACE the soda at a location in the target region and then to Look at it to verify the placement. A pre-condition of PLACE is that the location of the soda be known to within a loose tolerance, but one that is tighter than the tolerance in the initial belief state (note the large shadows in the initial belief). To tighten its estimate of the soda, the robot will need to LOOK at it. But, the soup is blocking the view of of the soda in the most likely state, so the plan moves the robot to a location in which the soda is likely to be visible. It then executes a LOOK, which sees both objects and reduces the uncertainty on the soda (and the soup) to the point that the planning can proceed. If it had failed to see the soda because of the blocking soup can, this would have triggered replanning, with the updated soup position

In figure 23(b) we see a more concrete version of the plan to place the soda, which involves a PICK followed by a PLACE. The pre-conditions of PICK require that the robot know the relative pose well enough, that it be holding the soda and that the swept volume of the approach path to the soda grasp point be believed to be clear except for the soda. However, the approach region is not in fact clear due to the presence of the soup. So, the robot constructs a plan to PLACE the soup at a pose in the warehouse region. Since the grasp approach region for the soup is clear, it MOVEs to the pre-grasp location. However, the move has increased the uncertainty in the soup's location, so a LOOK is added to the plan before the actual PICK. When planning the LOOK, the robot notes that its hand is in the way of seeing the soup, so the look motion involves a MOVE to a configuration where the hand is out of the way. The robot finally executes the LOOK, moving the head towards the soup and receiving an observation. However, at this point, the estimated pose of

[2]*Mens et Manus*, that is, "Mind and Hand" is the MIT motto and the name of our PR2.

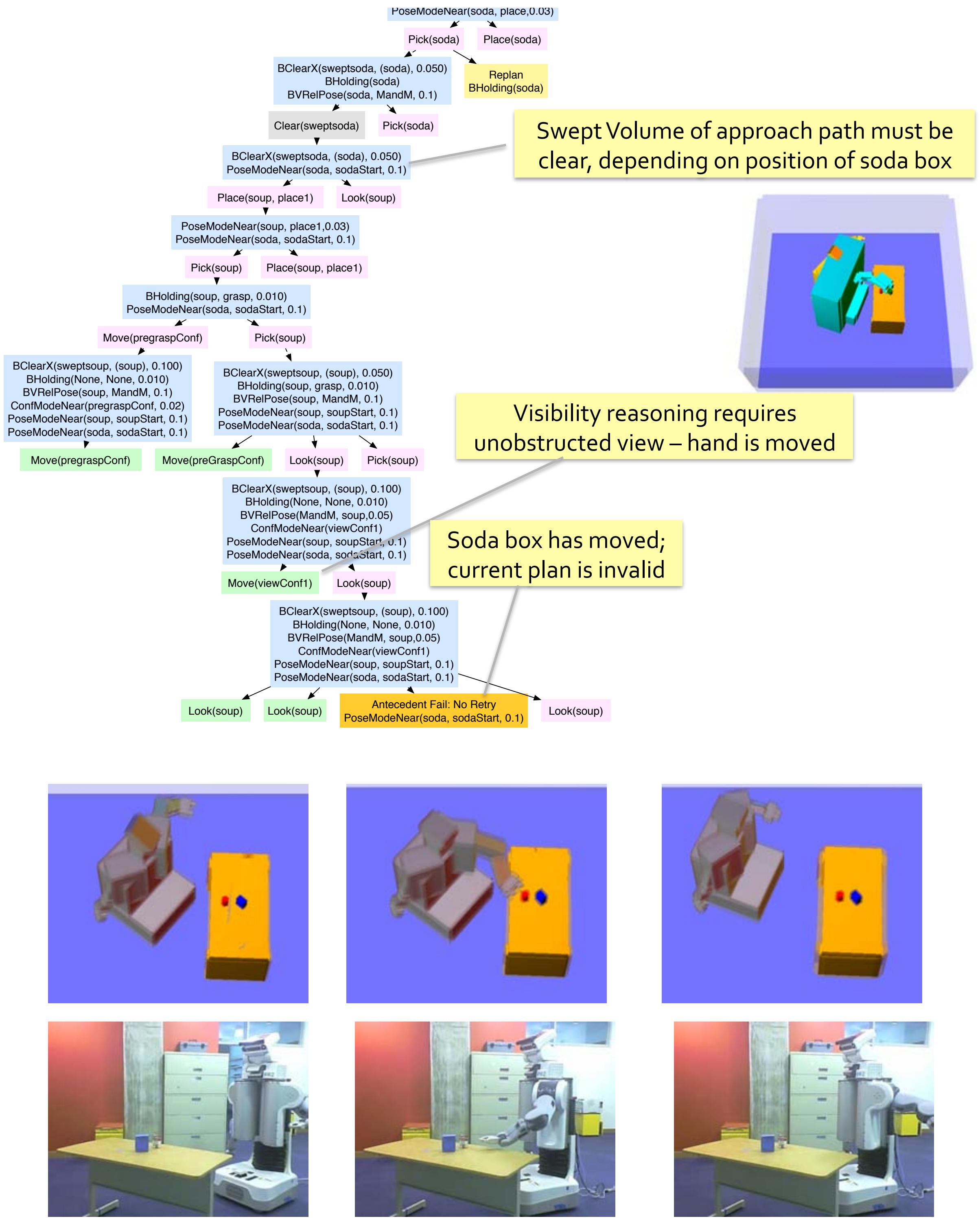


BIn(soda, target, 0.05)
Observe to verify
Place(soda, place)
Look(soda)
Gather information to enable detailed planning
PoseModeNear(soda, place,0.03)
Location in target chosen on the basis of clearance
Look(soda)
Place(soda, place)
BVRelPose(soda, MandM, 0.1)
Loose condition on knowledge of relative pose of soda and robot
Move(viewConf)
Look(soda)
Move(viewConf)
Visibility reasoning to choose viewConf



(a) Finding the soda box.

Figure 23: PR2 tabletop experiment


PoseModeNear(soda, place,0.03)
Pick(soda)
Place(soda)
BClearX(sweptsoda, (soda), 0.050) BHolding(soda) BVRelPose(soda, MandM, 0.1)
Replan BHolding(soda)
Clear(sweptsoda)
Pick(soda)
Swept Volume of approach path must be clear, depending on position of soda box
BClearX(sweptsoda, (soda), 0.050) PoseModeNear(soda, sodaStart, 0.1)
Place(soup, place1)
Look(soup)
PoseModeNear(soup, place1,0.03) PoseModeNear(soda, sodaStart, 0.1)
Pick(soup)
Place(soup, place1)
BHolding(soup, grasp, 0.010) PoseModeNear(soda, sodaStart, 0.1)
Move(pregraspConf)
Pick(soup)
BClearX(sweptsoup, (soup), 0.100) BHolding(None, None, 0.010) BVRelPose(soup, MandM, 0.1) ConfModeNear(pregraspConf, 0.02) PoseModeNear(soup, soupStart, 0.1) PoseModeNear(soda, sodaStart, 0.1)
BClearX(sweptsoup, (soup), 0.050) BHolding(soup, grasp, 0.010) BVRelPose(soup, MandM, 0.1) PoseModeNear(soup, soupStart, 0.1) PoseModeNear(soda, sodaStart, 0.1)
Visibility reasoning requires unobstructed view – hand is moved
Move(pregraspConf)
Move(preGraspConf)
Look(soup)
Pick(soup)
BClearX(sweptsoup, (soup), 0.100) BHolding(None, None, 0.010) BVRelPose(MandM, soup,0.05) ConfModeNear(viewConf1) PoseModeNear(soup, soupStart, 0.1) PoseModeNear(soda, sodaStart, 0.1)
Soda box has moved; current plan is invalid
Move(viewConf1)
Look(soup)
BClearX(sweptsoup, (soup), 0.100) BHolding(None, None, 0.010) BVRelPose(MandM, soup,0.05) ConfModeNear(viewConf1) PoseModeNear(soup, soupStart, 0.1) PoseModeNear(soda, sodaStart, 0.1)
Look(soup)
Look(soup)
Antecedent Fail: No Retry PoseModeNear(soda, sodaStart, 0.1)
Look(soup)

(b) Trying to move the red can out of the way.

Figure 23: PR2 tabletop experiment

the soda has moved sufficiently far from where it was thought to be when the plan was made that the current plan is deemed invalid at many levels of abstraction and replanning is initiated near the top of the tree (the "replan" node in yellow indicates where the new plan will be initiated).

In figure 23(c) we see the new plan for PICK and PLACE of the soda, which in turn requires a PICK and PLACE of the soup. This time, there are no surprises, since the robot base has not moved, so the robot can LOOK at soup and PICK it. Then, the robot moves to the chosen pose for placing the soup; it needs to localize itself relative to the target location, but because the location is a non-physical object, it is defined relative to the table, the robot LOOKs at the table twice to decrease its uncertainty relative to the table and then proceeds to PLACE the soup on the table.

In figure 23(d) we see the tree for picking up the soda. The robot checks that the swept volume for approaching the soda is still clear and MOVEs to the pre-grasp location. After the initial MOVE, the robot is not satisfied with its relative pose to the soda, so it does another small MOVE to correct its position. Then, the robot LOOKs at the soda again, which requires moving the hand out of the way. Finally, it PICKs up the soda.

In figure 24(a) we can see the final piece of the plan. The robot MOVEs to the location near the target place, LOOKs at the table and PLACEs the soda on the table. Since the MOVE was short and the original requirement on the target placement was not very stringent, the final uncertainty of the soda is acceptable and the final LOOK in the top-level plan does not have to be executed.

7.2 PR2 example with cupboard

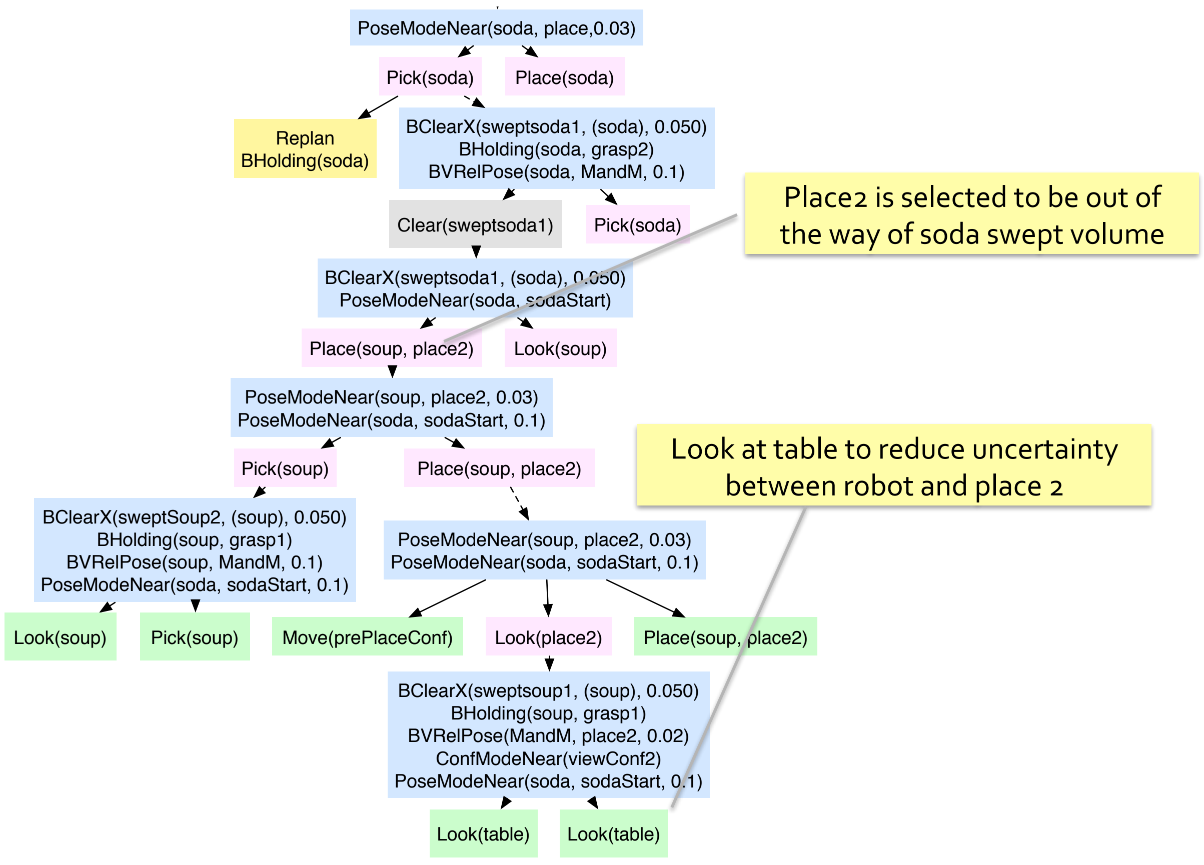
In this example, we gave the robot a repeated sequence of goals to place first the soda box and then the soup can into a region at the left end of the cupboard on the table. The robot executed this sequence several times.

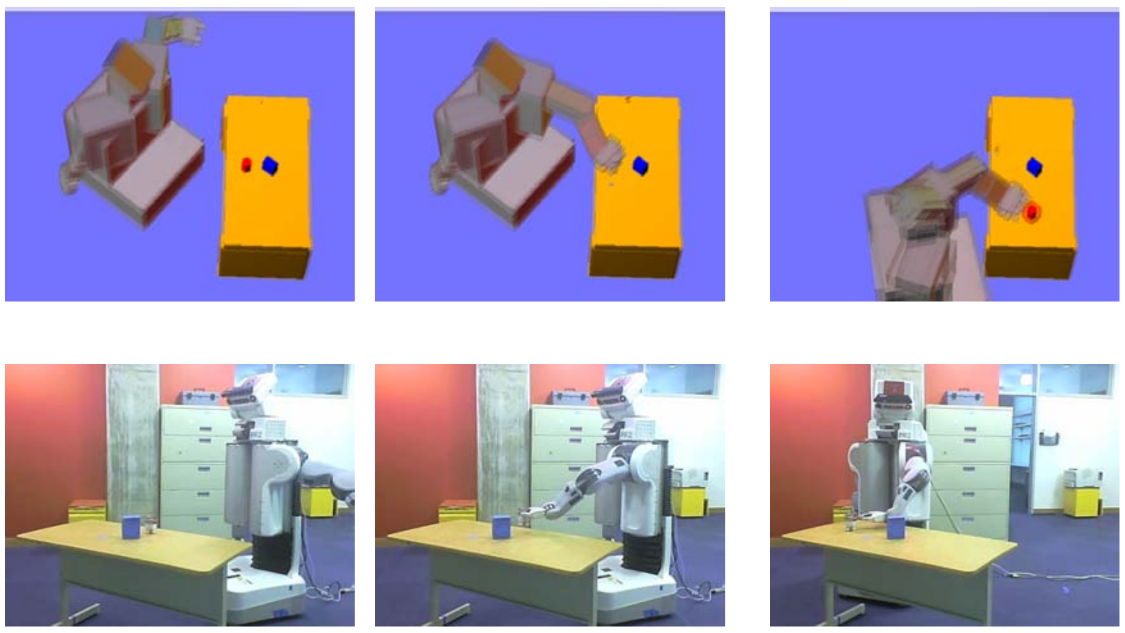
Figure 25(a) shows key frames from the belief state and the actual execution of the first goal: to place the soda in the target region from its initial position on the top shelf. This required moving the soup can out of the way (since it is blocking the target region) by PLACEing it on the table. Since the initial approach to the soup required a substantial base MOVE, the robot performs a LOOK before grasping, which requires MOVEing the hand out of the way. The subsequent approach to grasping the soda does not require moving the base, so the robot does not perform an additional LOOK.

Figure 25(b) shows a subsequent case of placing the soda in the target region. The soda and the soup start out in locations where they were placed by an earlier iteration of this process. When moving the can out of the way, a target location is chosen on the table which ends up being in the way of grasping the soda, so an additional motion is planned and executed to clear access to the soda box. During this subsequent planning step, the swept volume for accessing the soda is a constraint on the placement of the soup.

We have replicated this experiement quite a few times; it typically performs at least one or two cycles of moving one object out of the way so so to place the other object at the goal. The runs eventually fail in one of two ways:

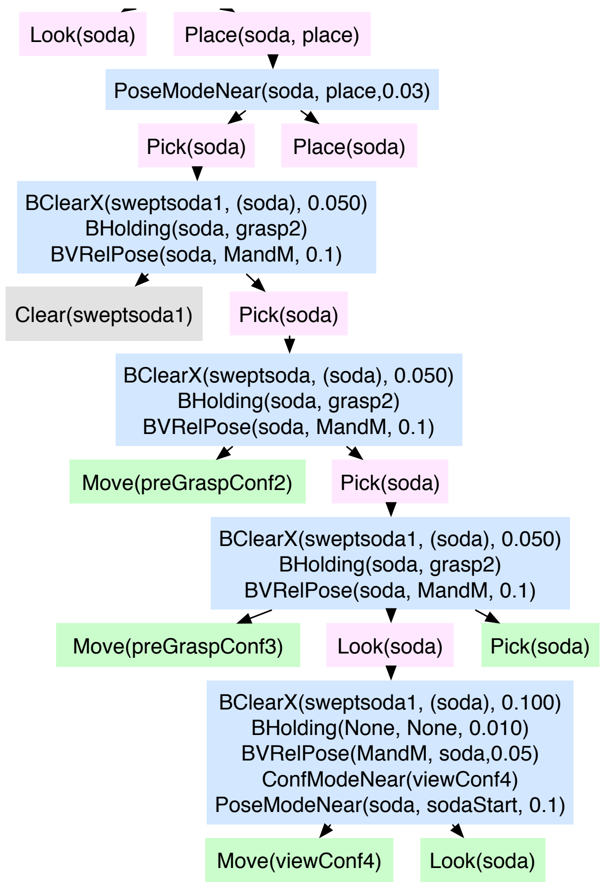
- The blue soda box slips within the fingers; it subsequently collides with the table or the cupboard since our model of where it is being held is very inaccurate. This suggests that we need to have a better grasp strategy and also a better model of the uncertainty in the grasps. One could then plan for actions that reduce the uncertainty either actively, by looking at the object in the hand, or passively, by exploting slipping during placement.

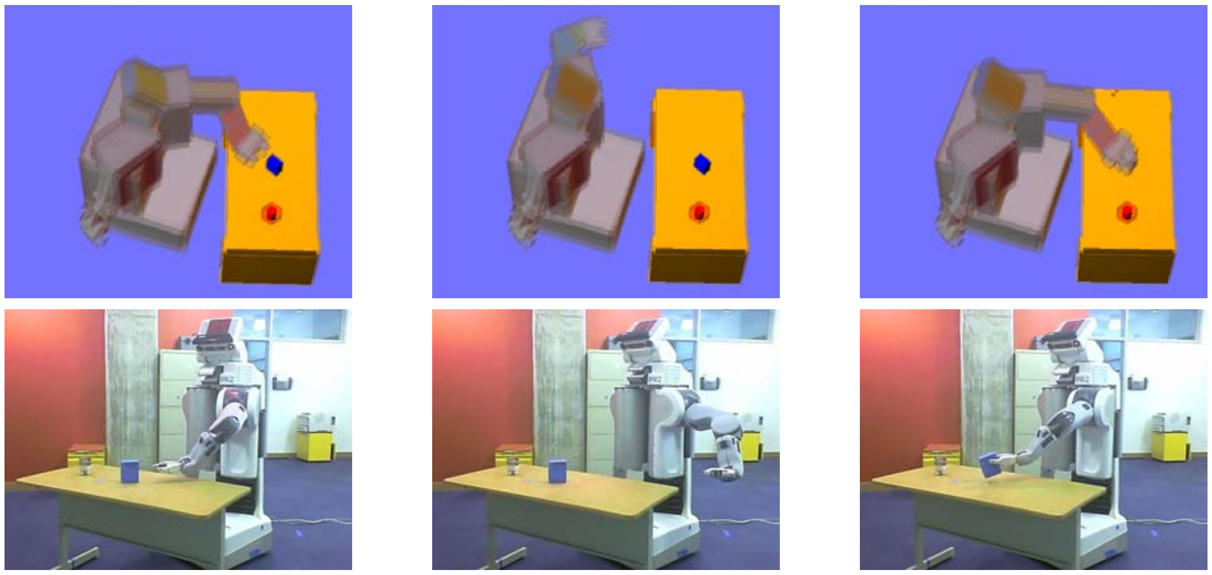

PoseModeNear(soda, place,0.03)
Pick(soda)
Place(soda)
Replan BHolding(soda)
BClearX(sweptsoda1, (soda), 0.050) BHolding(soda, grasp2) BVRelPose(soda, MandM, 0.1)
Clear(sweptsoda1)
Pick(soda)
Place2 is selected to be out of the way of soda swept volume
BClearX(sweptsoda1, (soda), 0.050) PoseModeNear(soda, sodaStart)
Place(soup, place2)
Look(soup)
PoseModeNear(soup, place2, 0.03) PoseModeNear(soda, sodaStart, 0.1)
Pick(soup)
Place(soup, place2)
Look at table to reduce uncertainty between robot and place 2
BClearX(sweptSoup2, (soup), 0.050) BHolding(soup, grasp1) BVRelPose(soup, MandM, 0.1) PoseModeNear(soda, sodaStart, 0.1)
PoseModeNear(soup, place2, 0.03) PoseModeNear(soda, sodaStart, 0.1)
Look(soup)
Pick(soup)
Move(prePlaceConf)
Look(place2)
Place(soup, place2)
BClearX(sweptsoup1, (soup), 0.050) BHolding(soup, grasp1) BVRelPose(MandM, place2, 0.02) ConfModeNear(viewConf2) PoseModeNear(soda, sodaStart, 0.1)
Look(table)
Look(table)



(c) Actually moving the red can.

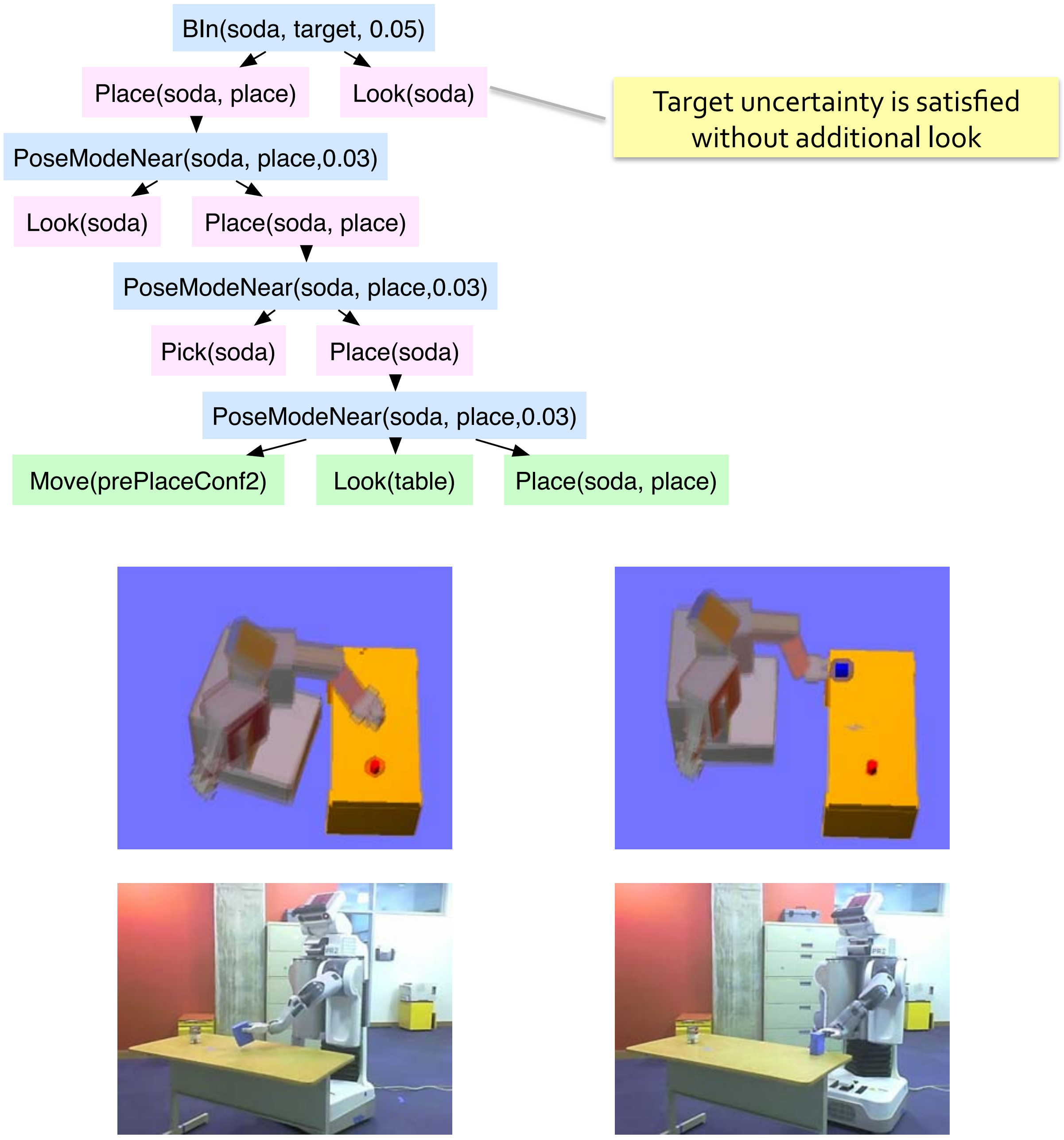
Figure 23: PR2 tabletop experiment


Look(soda)
Place(soda, place)
PoseModeNear(soda, place,0.03)
Pick(soda)
Place(soda)
BClearX(sweptsoda1, (soda), 0.050)
BHolding(soda, grasp2)
BVRelPose(soda, MandM, 0.1)
Clear(sweptsoda1)
Pick(soda)
BClearX(sweptsoda, (soda), 0.050)
BHolding(soda, grasp2)
BVRelPose(soda, MandM, 0.1)
Move(preGraspConf2)
Pick(soda)
BClearX(sweptsoda1, (soda), 0.050)
BHolding(soda, grasp2)
BVRelPose(soda, MandM, 0.1)
Move(preGraspConf3)
Look(soda)
Pick(soda)
BClearX(sweptsoda1, (soda), 0.100)
BHolding(None, None, 0.010)
BVRelPose(MandM, soda,0.05)
ConfModeNear(viewConf4)
PoseModeNear(soda, sodaStart, 0.1)
Move(viewConf4)
Look(soda)



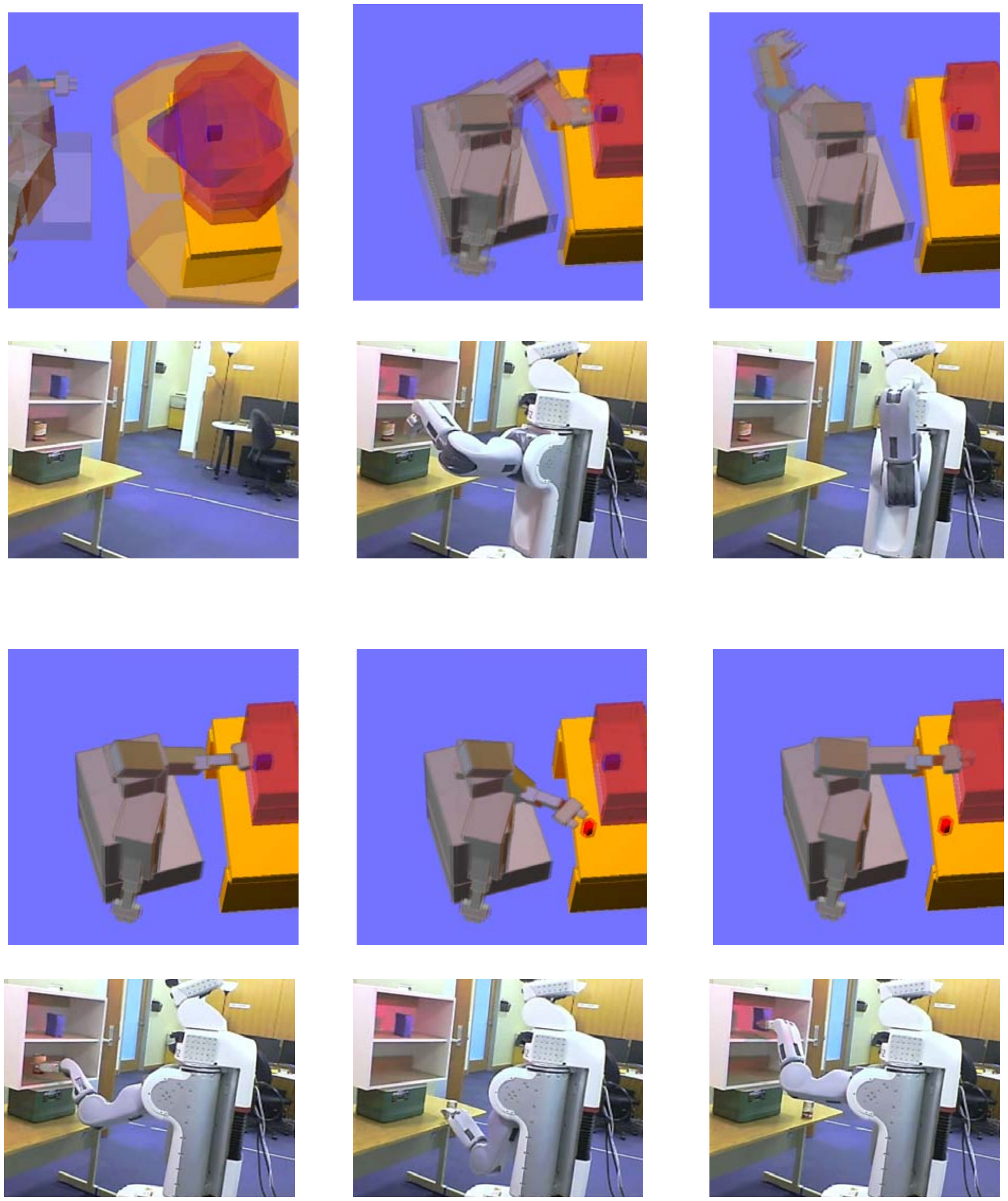
(d) Picking up the soda box.

Figure 23: PR2 tabletop experiment



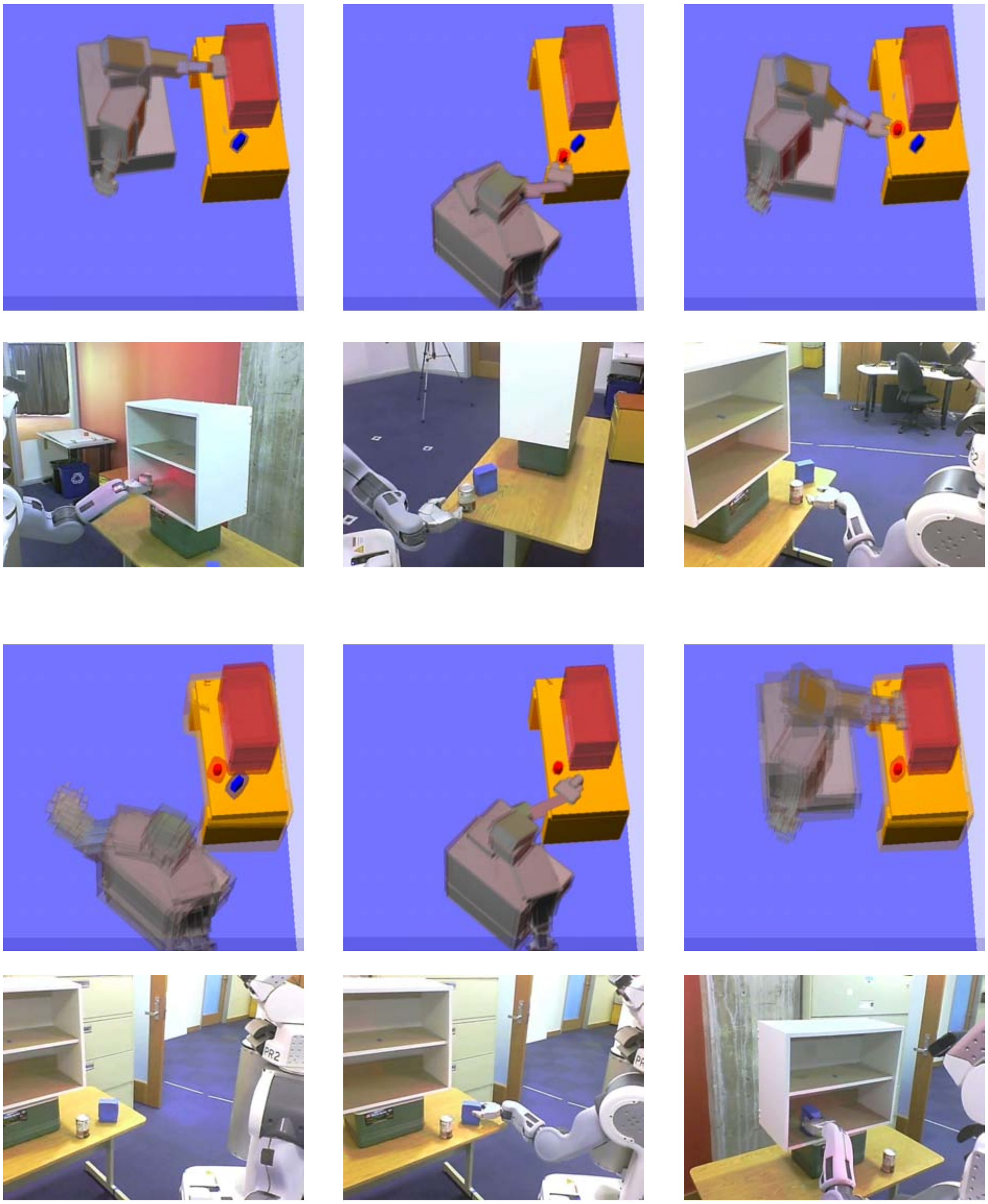
(a) Placing the soda box in the target region.

Figure 24: PR2 tabletop experiment



(a) Moving the soda box from the top shelf to a region on the bottom shelf blocked by the soda can. The initial conditions were set up by the authors.

Figure 25: PR2 cupboard experiment.



(b) Moving the soda box from the table to a region on the bottom shelf blocked by the red can. Note that the initial placement of the can on the table blocks access to the box and so it needs to be moved again. The initial conditions are the result of a previous sequence of robot moves.

Figure 25: PR2 cupboard experiment.

- Because of randomization in the path planner, the robot follows a very long path to a pick or a place; it builds up so much uncertainty that it collides with the cupboard or the table. We are replacing the current RRT implementation with an RRT* implementation which should reduce these problems. However, we should also never rely on performing long open-loop displacements of the base; they should be broken down into segments that re-localize the robot relative to relevant landmarks. Furthermore, the transition update in the UKF treats the control as the difference between the raw odometry poses before and after the motion, and assumes the variance in the motion is proportional to the magnitude of that difference; the update could be made more accurate by reporting the integrated motion distance of the robot and using variance proportional to that in the update.

7.3 Simulated examples using observed space oct-tree

Examples in this section illustrate the version of the planner that uses the observed space oct-tree; this is not yet implemented on the PR2.

7.3.1 Moving an object to enable looking

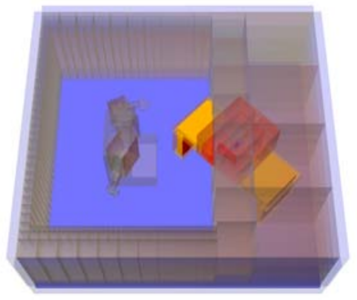
Figure 26 shows the sequence of belief states arising from a goal to know the pose of the blue cup accurately. In order to do so, the robot must first move the large occluding red object out of the way, and then move so it can observe the blue object. This example illustrates taking physical, non-sensing, actions in service of a purely informational goal. Note that this is in fact common when HPN plans in belief space, for example when the robot planned to move so as to be able to look at the soda in section 7.1.

7.3.2 Simple pick and place

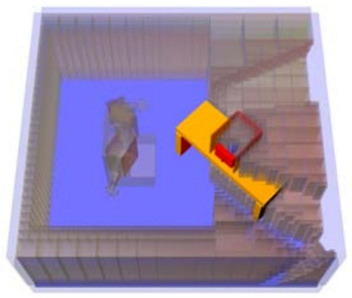
Figure 27 shows the sequence of belief states, including the observed space oct-tree, generated by a planning and execution run of BHPN. The goal is for the red object to be placed in a goal region on the side of the table; the uncertainty in placing the object is very high, so the first time the robot places the object, it is in fact not inside the goal region. Because the goal condition is $BIn(cupA, goalRegion, 0.001)$, the robot looks at the cup to verify its pose after placing it. In so doing, it discovers that the cup is not in the goal region, and replans, picking the cup up and re-placing it, then looking to verify and determining that the cup is very likely to be in the goal region, thereby satisfying the goal.

7.3.3 More complex example

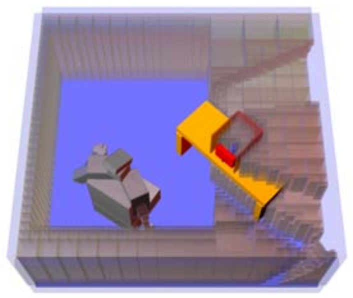
The next example is more complex, demonstrating longer chains of planning dependence. The robot is trying to place the blue cup where the green cup is currently located. It must move the green cup out of the way, then move the red cup, which is in the way of getting out the blue cup, and finally pick up the blue cup and place it. Throughout the course of this execution process, the robot interleaves look actions to reduce uncertainty and motions to improve visibility. There was no explicit attempt to program this particular ordering of actions; exactly how the execution unfolds depends on the particular observations that are received and on various random choices in both the high and low-level planning algorithms. Figure 28 shows a selection of the belief states generated during execution (the whole sequence is too long to include here.)



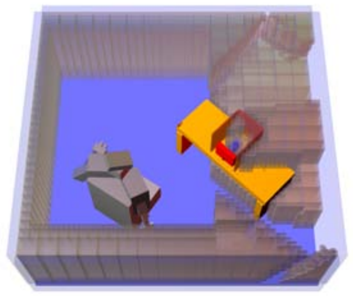
(a) Initial belief state.



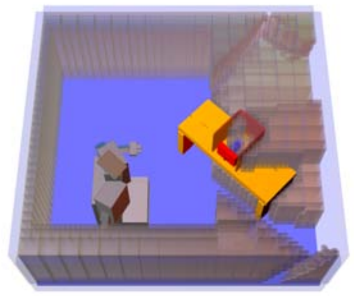
(b) After LOOK: table, cupboard and red object are well localized with respect to robot, but the blue object is occluded.



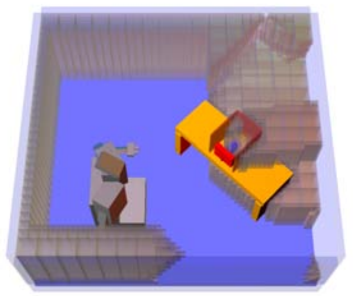
(c) After MOVE: robot moves to try to get a view of the blue object; its pose uncertainty is increased.



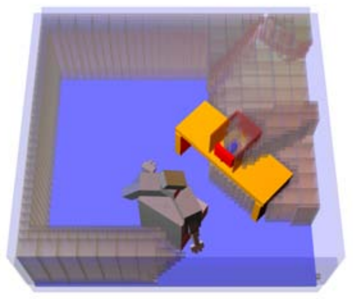
(d) After LOOK: now the large red object is well localized with respect to the robot and recognized to be an occlusion to viewing the blue object.



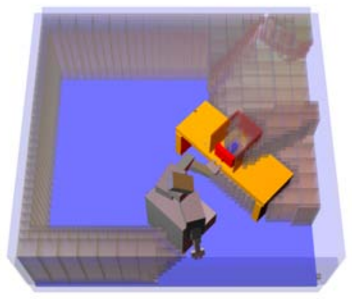
(e) After MOVE: rotate to look at unknown region that must be traversed to reach grasp for red object. red object.



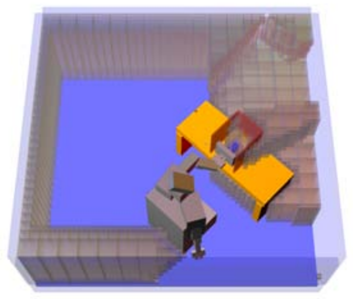
(f) After LOOK: the approach region is found to be clear.



(g) After MOVE and LOOK: the target region for placing the red object in the warehouse is found to be clear.

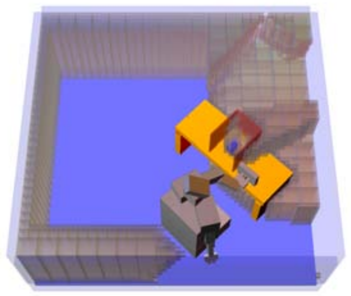


(h) After MOVE: robot configuration mean is at pre-grasp configuration.

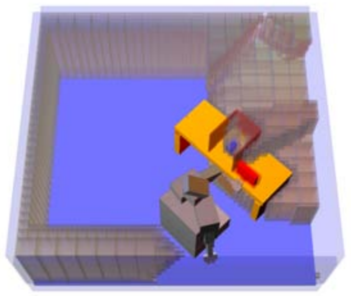


(i) After PICK: robot is holding the red object.

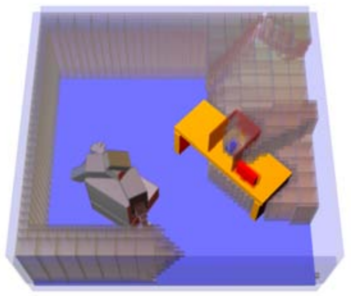
Figure 26: Moving an object to enable looking.



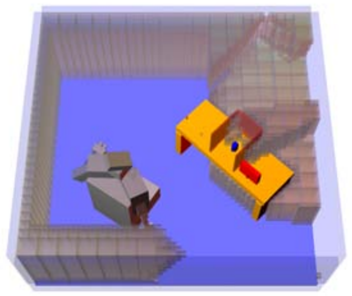
(j) After Move: robot is in pre-place pose and has low pose uncertainty because it did not move far.



(k) After Place: red cup is placed with fairly high confidence in the warehouse.

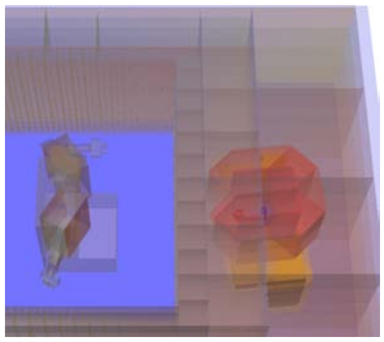


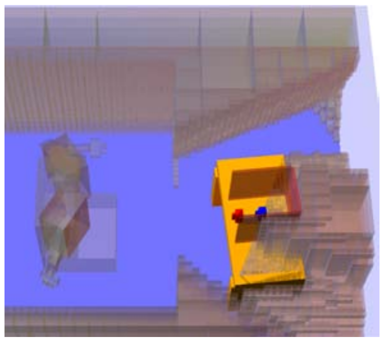
(l) After Move: robot is in position for an unoccluded look at the blue object.

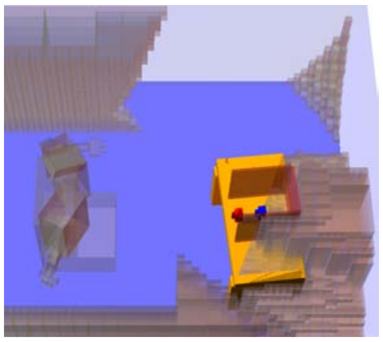


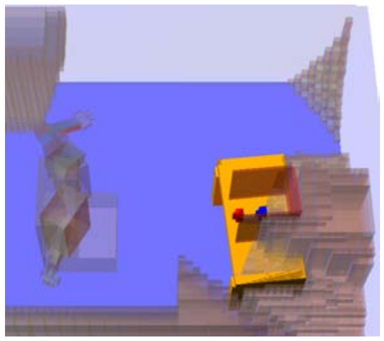
(m) After Look: blue object is well localized with respect to the robot and the goal is satisfied.

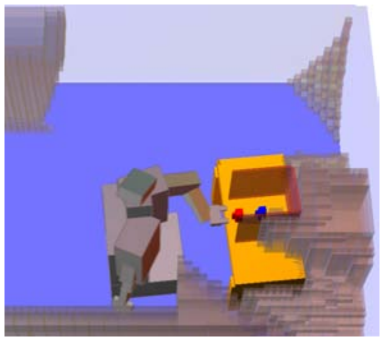
Figure 26: Moving an object to enable looking.

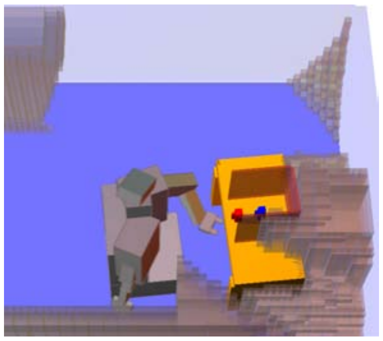

(a) Initial belief state.

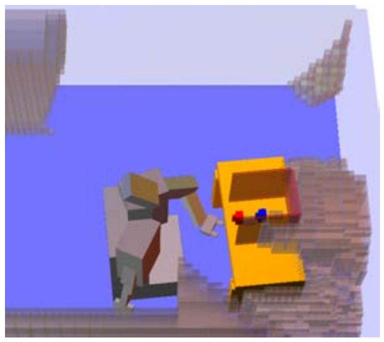

(b) After Look: table, cupboard, and red cup become visible.

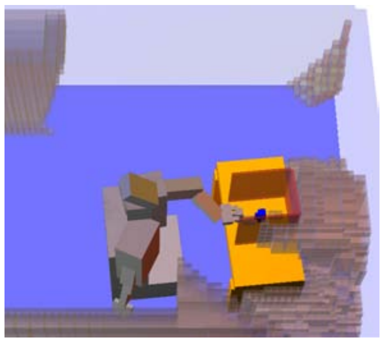

(c) After Look: area to the side of the table is partly visible.


(d) After Look: additional area to the side of the table is now visible.


(e) After Move: robot is ready to grasp red cup.


(f) Robot moves hand to offer unoccluded view of red cup.


(g) After Look: red cup is well localized with respect to robot.


(h) After Pick: robot is holding red cup.

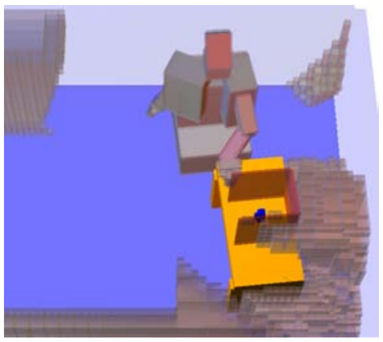
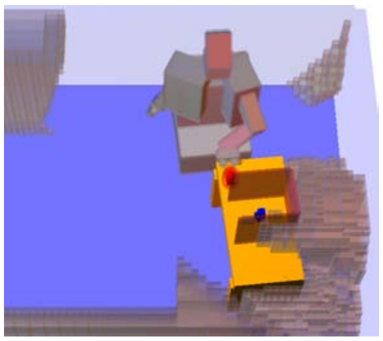
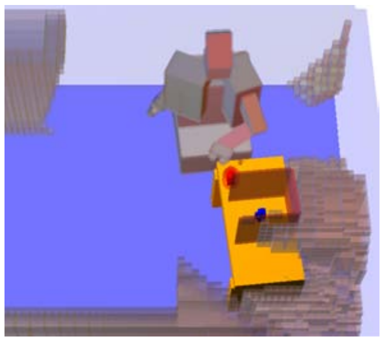

(i) After Move: mode of robot pose is in the pre-place configuration, but uncertainty is high.

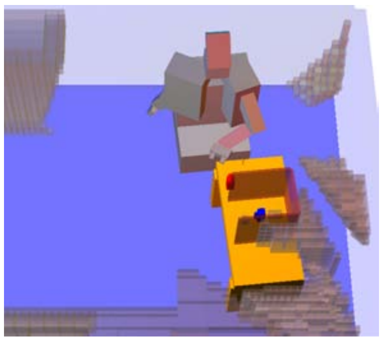
Figure 27: Simple pick and place example with observed space oct-tree.



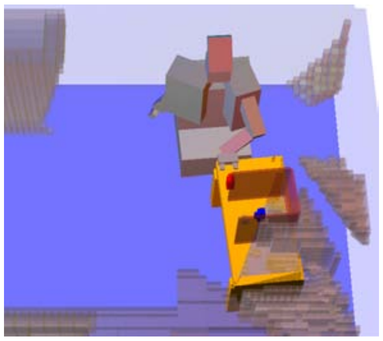
(j) After Place: mode of the red cup's pose is satisfactory, but uncertainty is high due to variance in the placing process.



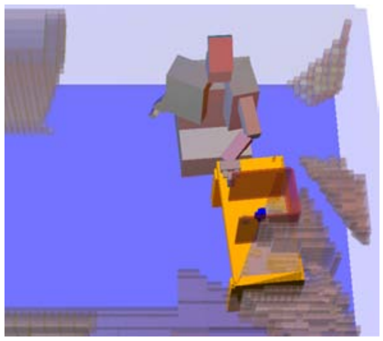
(k) Robot moves hand to offer unoccluded view of red cup.



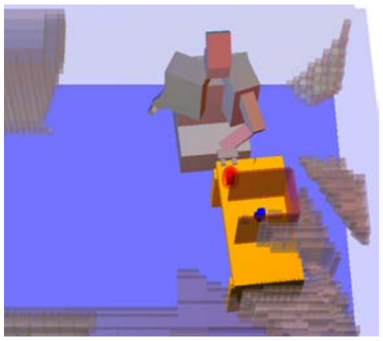
(l) after Look at the red cup: it is well localized, but the mode is not correct, so the goal condition is not satisfied. At this point, the replanning is initiated at several levels of the hierarchy, and the robot prepares to pick the object up and place it again.



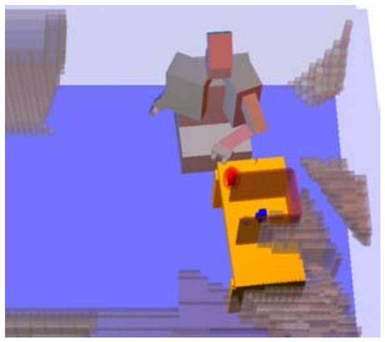
(m) After Move: to put the robot in position to pick and place the object again.



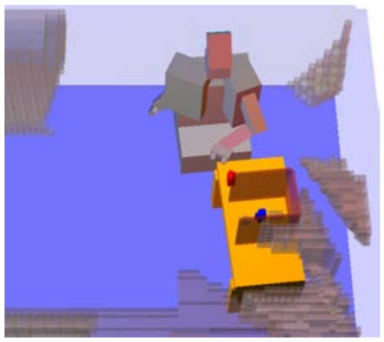
(n) After Pick: robot is holding red cup.



(o) After Place: mode of the red cup's pose is satisfactory, but uncertainty is high due to variance in the placing process.

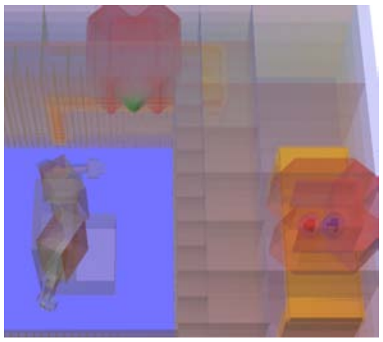


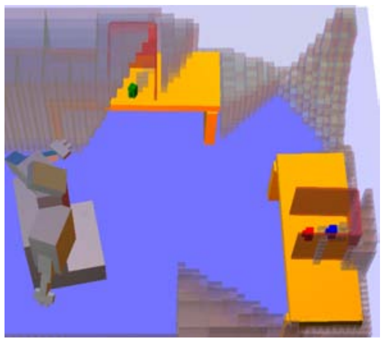
(p) Robot moves hand to offer unoccluded view of red cup. After the Look, the mode is still not correct, so the Move, Pick, Place, Look process is repeated.

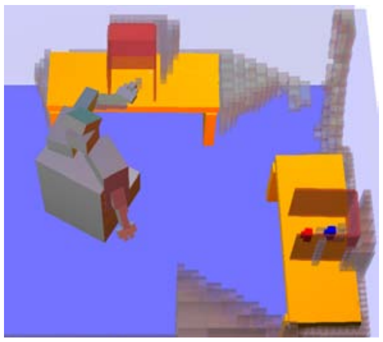


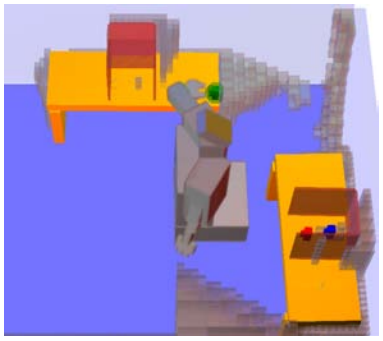
(q) Final belief state satisfies goal conditions: red cup is believed, with high probability, to be within the goal region.

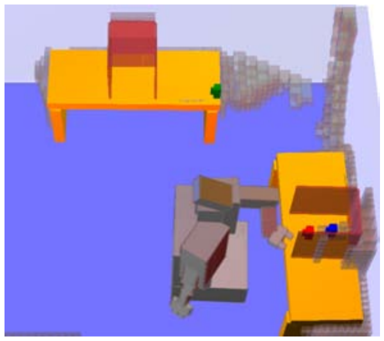
Figure 27: Simple pick and place example with observed space oct tree.

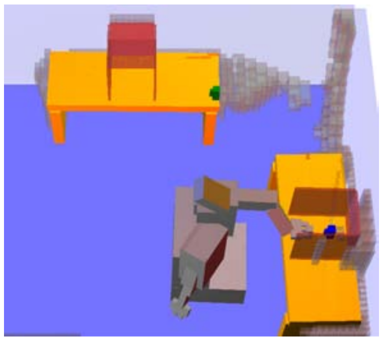

(a) Initial belief state.

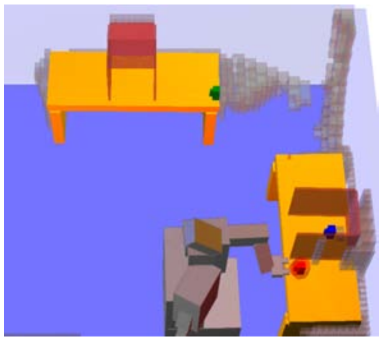

(b) After two LOOK operations to ensure that swept volumes are clear.

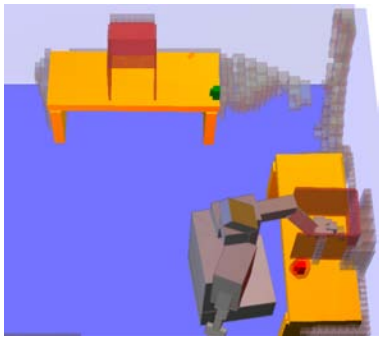

(c) Picking up the green object after one more LOOK operations to ensure the way is clear and a LOOK to localize the green object.


(d) Placing the green object out of the way.


(e) Localizing the red object.


(f) Picking up the red object.


(g) Placing the red object out of the way.


(h) Picking the blue object.

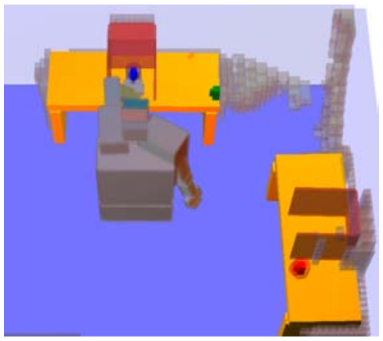

(i) Placing the blue object in the desired goal region.

Figure 28: Moving two objects out of the way using observed space oct-tree.

8 Conclusion

This paper has described a tightly integrated system that weaves together perception, estimation, geometric reasoning, symbolic task planning, and control to generate behavior in a real robot that robustly achieves tasks in uncertain domains. It is founded on these principles:

- Planning explicitly in the space of the robot's *beliefs* about the state of the world is necessary for intelligent information-gathering behavior;
- Interleaved hierarchical planning and execution fits beautifully with information gain: the system makes a high-level plan to gather information and then uses it, and the interleaved hierarchical planning and execution architecture ensures that planning that depends on the information naturally takes place after the information has been gathered; and
- Planning with simplified domain models is efficient and can be made robust by detecting execution failures and replanning online.

This integrated general-purpose mechanism current supports robust, flexible, solution of simple mobile manipulation problems in the current implementation, and we expect it to serve as a foundation for the solution of significantly more complex problems in the future.

Acknowledgments

We thank all the members of the LIS research group for their help in this project, and in particular: Jared Glover, Aaron Fryman, Will Grathwol, Sanja Popovic and Dawit Zewdie for developing the 3D perception subsystem; Lawson Wong for the UKF implementation; Ashwin Deshpande and Dylan Hadfield-Menell for help with hierarchical planning; Jennifer Barry for help in testing the planner and with PR2 software; and Caelan Reed Garrett for help with Cython.

This work was supported in part by the NSF under Grant No. 1117325. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR MURI grant N00014-09-1-1051, from AFOSR grant AOARD-104135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center. We thank Willow Garage for the use of the PR2 robot as part of the PR2 Beta Program.

References

R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *Proceedings of the Robotics Science and Systems Conference*, 2007.

Jennifer Barry, Kaijen Hsiao, Leslie Kaelbling, and Tomás Lozano-Pérez. Manipulation with multiple action types. In *International Symposium on Experimental Robotics*, 2012.

Jennifer L. Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. DetH*: Approximate hierarchical solution of large Markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011.

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 473–478, 2001.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

Avrim Blum and John Langford. Probabilistic planning in the graphplan framework. In *European Conference on Planning*, pages 319–332, 1999.

Ronen I. Brafman, Jean claude Latombe, Yoram Moses, and Yoav Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *Journal of the ACM*, 44:668, 1997.

Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28, 2009.

Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

Mehmet Dogar and Siddhartha Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, June 2012.

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

C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*, November 2009.

N. E. du Toit and J. W. Burdick. Robotic motion planning in dynamic, cluttered, uncertain environments. In *IEEE Conference on Robotics and Automation*, 2010.

A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22:46–57, June 1989.

Michael Erdmann. Using backprojection for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):240–271, 1994.

T. Erez and W. Smart. A scalable method for solving high-dimensional continuous POMDPs using local approximation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2010.

P. Thomas Fletcher, Sarang Joshi, Conglin Lu, and Stephen Pizer. Gaussian distributions on lie groups and their application to statistical shape analysis. In *Proceedings of Information Processing in Medical Imaging*, pages 450–462, 2003.

C. Fritz and S. A. McIlraith. Generating optimal plans in highly-dynamic domains. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2009a.

Christian Fritz and Sheila A. McIlraith. Computing robust plans in continuous domains. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2009b.

Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pages 1537–1540, 2008.

Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning: Theory and practice.* Elsevier, 2004.

Kris Hauser. Randomized belief-space replanning in partially-observable continuous spaces. In *Workshop on Algorithmic Foundations of Robotics*, 2010.

Kris Hauser and Victor Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research*, 30(6):676–698, 2011.

Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Grasping POMDPs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *IEEE Review*, 92(3), 2004.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation*, 2011.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in the now. Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2012. URL `http://hdl.handle.net/1721.1/71521`.

James J. Kuffner, Jr. and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics Science and Systems*, 2008.

Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *International Journal of Robotics Research*, 30(3): 308–323, 2011.

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

Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006. URL `msl.cs.uiuc.edu/planning`.

Martin Levihn, Jonathan Scholz, and Mike Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Proceedings of the Workshp on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

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.

Bhaskara Marthi, Stuart Russell, and Jason Wolfe. Combined task and motion planning for mobile manipulation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.

D.Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35(7):814–824, 1990.

John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*. Ablex Publishing Company, Norwood, New Jersey, 1985.

Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 867–874, 1987.

Nils J. Nilsson. Triangle tables: A proposal for a robot programming language. Technical Report 347, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

Sylvie C. W. Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *International Journal of Robotics Research*, 29(8): 1053–1068, 2010.

Dejan Pangercic, Moritz Tenorth, Dominik Jain, and Michael Beetz. Combining perception and knowledge for everyday manipulation. In *IEEE Conference on Intelligent Robots and Systems*, 2010.

R. P. A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *International Conference on Automated Planning and Scheduling*, 2004.

Erion Plaku and Gregory Hager. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2010.

Robert Platt, Russell Tedrake, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. *International Symposium on Robotics Research*, 2007.

Radu Bogdan Rusu, Ioan Alexandru Sucan, Brian P. Gerkey, Sachin Chitta, Michael Beetz, and Lydia E. Kavraki. Real-time perception-guided motion planning for a personal robot. In *IEEE Conference on Intelligent Robots and Systems*, St. Louis, MO, 2009.

Scott Sanner and Kristian Kersting. Symbolic dynamic programming for first-order POMDPs. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2010.

Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research*, 23(7–8):729–746, 2004.

Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.

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

Anderson Souza, Andre Santana, Ricardo Britto, Luiz Gonalves, and Aderlardo Medeiros. Representation of odometry errors on occupancy grids. In *International Conference on Informatics in Control, Automation and Robotics*, 2008.

Siddharth Srinivasa, Dave Ferguson, Casey Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. HERB: A home exploring robotic butler. *Journal of Autonomous Robots*, 2009.

Mike Stilman and James J. Kuffner. Planning among movable obstacles with artificial constraints. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2006.

Michael Thielscher. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4-5):533–565, 2005.

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

Le-Chi Tuan, Chitta Baral, and Tran Cao Son. A state-based regression formulation for domains with sensing actions and incomplete information. *Logical Methods in Computer Science*, 2(4), 2006.

Jur van den Berg, Pieter Abbeel, and Kenneth Y. Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research*, 30(7):895–913, 2011.

Chenggang Wang and Roni Khardon. Relational partially observable MDPs. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2010.

Lawson L. S. Wong, Leslie P. Kaelbling, and Tomas Lozano-Perez. Collision-free state estimation. In *IEEE Conference on Robotics and Automation*, 2012.

K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010.

S. W. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. In *International Conference on Automated Planning and Scheduling*, 2007.

A Inferential methods

A.1 Entailment and contradiction for discrete domain

Here are the entailment and contradiction relations for the fluents describing the domain in section 5.1.

This table shows the conditions under which f_1 entails f_2, in a domain with n locations. A blank entry indicates no entailment is possible.

	entails f_2		
f_1	$MLLoc(l_2)$	$BLoc(l_2, \epsilon_2)$	$BVLoc(\epsilon_2)$
$MLLoc(l_1)$	$l_1 = l_2$		
$BLoc(l_1, \epsilon_1)$	$l_1 = l_2$ & $\epsilon_1 > 1/n$	$l_1 = l_2$ & $\epsilon_1 \leq \epsilon_2$	$\epsilon_1 \leq \epsilon_2$
$BVLoc(\epsilon_1)$			$\epsilon_1 \leq \epsilon_2$

This table indicates conditions under which f_1 contradicts f_2. It is symmetric, so entries in lower triangle are not repeated.

	contradicts f_2		
f_1	$MLLoc(l_2)$	$BLoc(l_2, \epsilon_2)$	$BVLoc(\epsilon_2)$
$MLLoc(l_1)$	$l_1 \neq l_2$	$(l_1 = l_2$ & $\epsilon_2 < 1/n) \mid\mid$ $(l_1 \neq l_2$ & $\epsilon_2 > 1/n)$	
$BLoc(l_1, \epsilon_1)$	symmetric	$(l_1 \neq l_2$ & $(1 - \epsilon_1) + (1 - \epsilon_2) > 1)$	
$BVLoc(\epsilon_1)$			

A.2 Inferential methods for pick and place domain

This is probably not a complete characterization of the contradiction and entailment relationships among th fluents, but they are sufficient to enable the current implementation to operate effectively.

A.3 Entailment

In the following conditions a vector $x \leq y$ if $x_i \leq y_i$ for all components x_i, y_i of x and y.

- $BVRelPose(o_{1a}, o_{2a}, \epsilon_a, \delta_a)$ entails $BVRelPose(o_{1b}, o_{2b}, \epsilon_b, \delta_b)$ if
$$o_{1a} = o_{1b} \;\&\; o_{2a} = o_{2b} \;\&\; \epsilon_a \leq \epsilon_b \;\&\; \delta_a \leq \delta_b \ \ ,$$

- $PoseModeNear(o_a, p_a, \delta_a)$ entails $PoseModeNear(o_b, p_b, \delta_b)$ if
$$o_a = o_b \;\&\; \delta_a \leq \delta_b \;\&\; |p_a - p_b| \leq \delta_b - \delta_a \ \ ,$$
where $|p_a - p_b|$ is a vector of absolute differences in the components of the pose.

- $ConfModeNear(c_a, \delta_a)$ entails $ConfModeNear(c_b, \delta_b)$ if
$$\delta_a \leq \delta_b \;\&\; |base(c_a) - base(c_b)| \leq \delta_b - \delta_a \;\&\; |hand(c_a) - hand(c_b)| \leq \delta_b - \delta_a \;\&\; grip(c_a) = grip(c_b) \ \ .$$

- $ConfModeNear(c_a, \delta_a)$ entails $PoseModeNear(\mathbf{robot}, p, \delta_b)$ if
$$\delta_a \leq \delta_b \;\&\; |base(c_a) - p| \leq \delta_b - \delta_a \ \ .$$

- $\mathit{BIn}(o_a, r_a, \epsilon_a)$ entails $\mathit{BIn}(o_b, r_b, \epsilon_b)$ if

$$o_a = o_b \;\&\; \mathit{contains}(o_b, o_a) \;\&\; \epsilon_a \leq \epsilon_b$$

A.4 Contradiction

Note that these relations are symmetric.

- $\mathit{PoseModeNear}(o_a, p_a, \delta_a)$ contradicts $\mathit{PoseModeNear}(o_b, p_b, \delta_b)$ if

$$o_a = o_b \;\&\; |p_a - p_b| > \delta_a + \delta_b$$

- $\mathit{PoseModeNear}(o_a, p_a, \delta_a)$ contradicts $\mathit{BIn}(o_b, r_b, \epsilon_b)$ if

$$o_a = o_b \;\&\; \neg \mathit{contains}(r_b, \mathit{vol}(o_a, p_a)) \;\;,$$

  where $\mathit{vol}(o, p)$ denotes the volume of object o when placed at pose p.

- $\mathit{ConfModeNear}(c_a, \delta_a)$ contradicts $\mathit{ConfModeNear}(c_b, \delta_b)$ if

$$|\mathit{base}(p_a) - \mathit{base}(p_b)| > \delta_a + \delta_b \textbf{ or } |\mathit{hand}(p_a) - \mathit{hand}(p_b)| > \delta_a + \delta_b \textbf{ or } |\mathit{grip}(p_a) - \mathit{grip}(p_b)| > 2\delta_{grip}$$

- $\mathit{BHolding}(o_a, g_a, \epsilon_a)$ contradicts $\mathit{BHolding}(o_b, g_b, \epsilon_b)$ if

$$o_a \neg = o_b \textbf{ or } (o_a = o_b \;\&\; |g_a - g_b| < \delta_g) \;\; .$$

- $\mathit{BHolding}(o_a, g_a, \epsilon_a)$ contradicts $\mathit{BIn}(o_b, r_b, \epsilon_b)$ if

$$o_a = o_b \;\;,$$

  because an object can't simultaneously be in the robot's hand and placed at a stable pose.

- $\mathit{BIn}(o_a, r_a, \epsilon_a)$ contradicts $\mathit{BIn}(o_b, r_b, \epsilon_b)$ if

$$o_a = o_b \;\&\; \neg \mathit{fits}(o_a, \mathit{intersection}(r_a, r_b))$$

- $\mathit{BIn}(o_a, r_a, \epsilon_a)$ contradicts $\mathit{BClearX}(r_b, x_b, \epsilon_b)$ if

$$o_a \notin x_b \;\&\; \neg \mathit{fits}(o_a, r_a \;\, r_b) \;\; .$$

- $\mathit{BClearX}(r_a, x_a, \epsilon_a)$ contradicts $\mathit{PoseModeNear}(o_b, p_b, \delta_b)$ if

$$o_b \notin x_a \;\&\; \mathit{overlaps}(\mathit{vol}(o_b, p_b), r_a) \;\; .$$

B Proofs and derivations

B.1 Discrete regression

Proposition 1.

$$\mathit{moveRegress}(\epsilon) = \frac{\epsilon - p_{\mathit{fail}}}{1 - p_{\mathit{fail}}}$$

Proof. We find it by solving equation 1 for ϵ_r such that $b_i = 1 - \epsilon_r$. To guarantee the resulting fluent is true, we need to guarantee:

$$\begin{aligned}
b'_j &> 1 - \epsilon \\
(1 - \epsilon_r)(1 - p_{\mathit{fail}}) + b_j &> 1 - \epsilon \\
\epsilon_r &< \frac{\epsilon - p_{\mathit{fail}} + b_j}{1 - p_{\mathit{fail}}}
\end{aligned}$$

For simplicity in the regression, we strengthen the condition, by treating b_j as zero, letting

$$\epsilon_r = \mathit{moveRegress}(\epsilon) = \frac{\epsilon - p_{\mathit{fail}}}{1 - p_{\mathit{fail}}}$$

□

Proposition 2.

$$\mathit{lookPosRegress}(\epsilon) = \frac{\epsilon(1 - p_{\mathit{fn}})}{\epsilon(1 - p_{\mathit{fn}}) + p_{\mathit{fp}}(1 - \epsilon)} \quad .$$

Proof. In order to guarantee that $\mathit{BLoc}(l_{\mathit{interest}}, \epsilon)$ holds after looking and seeing the object, we need to guarantee that $\mathit{BLoc}(l_{\mathit{interest}}, \epsilon_r)$ holds before, where $\epsilon_r = \mathit{lookPosRegress}(\epsilon)$.

$$\begin{aligned}
b'_i &= \Pr_{b'}(S = l_{\mathit{interest}} \mid A = \textsc{look}(l_{\mathit{interest}}), O = \mathit{true}) \\
b'_i &= \frac{\Pr(O = \mathit{true} \mid S = l_{\mathit{interest}}, A = \textsc{look}(l_{\mathit{interest}}))\Pr_b(S = l_{\mathit{interest}})}{\Pr(O = \mathit{true} \mid A = \textsc{look}(l_{\mathit{interest}}))} \\
b'_i &= \frac{(1 - p_{\mathit{fn}})b_i}{(1 - p_{\mathit{fn}})b_i + p_{\mathit{fp}}(1 - b_i)} \\
1 - \epsilon &= \frac{(1 - p_{\mathit{fn}})(1 - \epsilon_r)}{(1 - p_{\mathit{fn}})(1 - \epsilon_r) + p_{\mathit{fp}}\epsilon_r} \\
\epsilon_r &= \frac{\epsilon(1 - p_{\mathit{fn}})}{\epsilon(1 - p_{\mathit{fn}}) + p_{\mathit{fp}}(1 - \epsilon)}
\end{aligned}$$

□

Proposition 3.

$$\mathit{lookNegRegress}(\epsilon) = \frac{(1 - p_{\mathit{fp}}) - (1 - \epsilon_i)(p_{\mathit{fn}}(1 - \epsilon_j) + \epsilon_j(1 - p_{\mathit{fp}}))}{1 - p_{\mathit{fp}}} \quad .$$

Proof. In order to guarantee that $BLoc(l_{interest}, \epsilon)$ after looking at location l_{target} and not seeing the object, we need to guarantee that $BLoc(l_{interest}, \epsilon_r)$ holds before, where $\epsilon_r = lookNegRegress(\epsilon_{interest}, \epsilon_{target})$.

$$\begin{aligned}
b'_i &= \Pr_{b'}(S = l_{interest} \mid A = \text{LOOK}(l_{target}), O = \mathit{false}) \\
b'_i &= \frac{\Pr(O = \mathit{false} \mid S = l_{interest}, A = \text{LOOK}(l_{target}))\Pr_b(S = l_{interest})}{\Pr(O = \mathit{true} \mid A = \text{LOOK}(l_{interest}))} \\
b'_i &= \frac{(1 - p_{fp})b_i}{p_{fn}b_j + (1 - p_{fp})(1 - b_j)} \\
1 - \epsilon_i &= \frac{(1 - p_{fp})(1 - \epsilon_r)}{p_{fn}(1 - \epsilon_j) + (1 - p_{fp})\epsilon_j} \\
\epsilon_r &= \frac{(1 - p_{fp}) - (1 - \epsilon_i)(p_{fn}(1 - \epsilon_j) + \epsilon_j(1 - p_{fp}))}{1 - p_{fp}}
\end{aligned}$$

□

B.2 Continuous regression

Proposition 4.

$$obsRegress(\epsilon, \delta, \sigma_o) = 1 - \text{erf}\left(\sqrt{\text{erf}^{-1}(1 - \epsilon)^2 - \frac{\delta^2}{2\sigma_o^2}}\right) .$$

Proof. Our goal is to find $\epsilon_r = obsRegress(\epsilon, \delta, \sigma_o)$ such that if

$$\text{erf}\left(\frac{\delta}{\sqrt{2}\sigma_r}\right) < 1 - \epsilon_r \qquad (1)$$

before the observation, then

$$\text{erf}\left(\frac{\delta}{\sqrt{2}\sigma}\right) < 1 - \epsilon \qquad (2)$$

after the observation. If, before the observation, the belief is $\mathcal{N}(\mu_r, \sigma_r^2)$, then after an observation o, the belief will be

$$\mathcal{N}\left(\frac{\mu_r\sigma_o^2 + o\sigma_r^2}{\sigma_o^2 + \sigma_r^2}, \frac{\sigma_o^2\sigma_r^2}{\sigma_o^2 + \sigma_t^2}\right) .$$

so

$$\sigma^2 = \frac{\sigma_o^2\sigma_r^2}{\sigma_o^2 + \sigma_t^2} . \qquad (3)$$

Solving equations 1, 2, and 3 for ϵ_r in terms of ϵ, we get

$$\epsilon_r = obsRegress(\epsilon, \delta, \sigma_o) = 1 - \text{erf}\left(\sqrt{\text{erf}^{-1}(1 - \epsilon)^2 - \frac{\delta^2}{2\sigma_o^2}}\right) .$$

□

Proposition 5.

$$\mathit{changeRegress}(\epsilon, \delta, \sigma_Y) = 1 - \operatorname{erf}\left(\frac{\delta \operatorname{erf}^{-1}(1-\epsilon)}{\sqrt{\delta^2 - 2\sigma_Y^2 \operatorname{erf}^{-1}(1-\epsilon)^2}}\right) \quad .$$

Proof. In this case, the update equation for the variance is simply

$$\sigma^2 = \sigma_r^2 + \sigma_Y^2 \tag{4}$$

So, solving equations 1, 2, and 4 for ϵ_r in terms of ϵ, we get

$$\epsilon_r = \mathit{changeRegress}(\epsilon, \delta, \sigma_Y) = 1 - \operatorname{erf}\left(\frac{\delta \operatorname{erf}^{-1}(1-\epsilon)}{\sqrt{\delta^2 - 2\sigma_Y^2 \operatorname{erf}^{-1}(1-\epsilon)^2}}\right) \quad .$$

□

Proposition 6.

$$\mathit{probModeMoved}(\epsilon, \delta_b, v, \delta) = 2\Phi\left(\frac{\delta\sqrt{\sigma_r^2 + \sigma_o^2}}{2\sigma_r^2}\right) \quad .$$

Proof. Letting the distribution before the observation be $\mathcal{N}(\mu_r, \sigma_r^2)$, the posterior distribution after making an observation x_o with variance σ_o^2 is $\mathcal{N}(\mu, \sigma_o^2)$, where

$$\mu = \frac{\sigma_o^2 \mu_r + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} \quad .$$

The probability that $\mathit{ModeNear}(v, \delta)$ is violated after the observation is the probability that μ is not in the interval $(v - \delta, v + \delta)$. That is

$$p_{\mathit{fail}} = \int_x \Pr(\mu_{x_r} = x)\left(\Pr(\mu_x < v - \delta \mid \mu_{x_r} = x) + \Pr(\mu_x > v + \delta \mid \mu_{x_r} = x)\right) \quad ,$$

which can be written in terms of the prior distribution and the observation as

$$p_{\mathit{fail}} = \int_x \Pr(\mu_{x_r} = x)\left(\Pr\left(\frac{\sigma_o^2 x + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} < v - \delta\right) + \Pr\left(\frac{\sigma_o^2 x + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} > v + \delta\right)\right) \quad .$$

We will now concentrate on the first term of the sum, where the random variable of interest is x_o (all the other quantities are known).

$$\Pr\left(\frac{\sigma_o^2 x + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} < v - \delta\right) \quad = \quad \Pr\left(x_o < \frac{(\sigma_r^2 + \sigma_o^2)(v - \delta) - \sigma_o^2 x}{\sigma_r^2}\right)$$

The observation x_o was drawn from distribution $\mathcal{N}(X, \sigma_o^2)$ where X is the (unknown) true value of X; combining that with the prior distribution on X, we have

$$\Pr(x_o) = \int_x \Pr(X = x) \Pr(x_o \mid X = x) \quad .$$

This combination of Gaussian distributions is also Gaussian, so the distribution on x_o is $\mathcal{N}(\mu_{x_r}, \sigma_o^2 + \sigma_r^2)$. So,

$$\Pr\left(\frac{\sigma_o^2 x + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} < v - \delta\right) \quad = \quad \Phi\left(\frac{\sqrt{\sigma_r^2 + \sigma_o^2}(x - v + \delta)}{\sigma_r^2}\right)$$

Where Φ is the Gaussian CDF. By a similar argument,

$$\Pr\left(\frac{\sigma_o^2 x + \sigma_r^2 x_o}{\sigma_o^2 + \sigma_r^2} > v + \delta\right) \quad = \quad 1 - \Phi\left(\frac{\sqrt{\sigma_r^2 + \sigma_o^2}(x - v - \delta)}{\sigma_r^2}\right)$$

We can rewrite the failure probability as:

$$p_{fail} = \int_x \Pr(\mu_{x_r} = x)\left(\Phi\left(\frac{\sqrt{\sigma_r^2 + \sigma_o^2}(x - v + \delta)}{\sigma_r^2}\right) + 1 - \Phi\left(\frac{\sqrt{\sigma_r^2 + \sigma_o^2}(x - v - \delta)}{\sigma_r^2}\right)\right) \quad .$$

We know $\mu_{x_r} \in v \pm \delta$ but do not have any further distributional information, so we will assume it is uniform. The integral is still too hard to evaluate, so we will evaluate the first term of the sum at $x = v - \delta/2$ and the second term at $x = v + \delta/2$; this gives us the probability that the new mean will be below the interval given that the old mean was at the $25th$ percentile plus the probability that the new mean will be above the interval given that the old mean was at the $75th$ percentile. The resulting approximation is

$$\begin{aligned}\hat{p}_{fail} &= \Phi\left(\delta\frac{\sqrt{\sigma_r^2 + \sigma_o^2}}{2\sigma_r^2}\right) + 1 - \Phi\left(-\delta\frac{\sqrt{\sigma_r^2 + \sigma_o^2}}{2\sigma_r^2}\right) \\ &= 2\Phi\left(\frac{\delta\sqrt{\sigma_r^2 + \sigma_o^2}}{2\sigma_r^2}\right)\end{aligned}$$

$\square$