Integrated 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

We describe an integrated strategy for planning, perception, state-estimation and action in complex mobile manipulation domains based on planning in the *belief space* of probability distributions over states using hierarchical goal regression (pre-image back-chaining). We develop a vocabulary of logical expressions 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 can give rise 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 combines geometric and symbolic representations in an aggressively hierarchical planning architecture, called HPN (for *hierarchical planning in the now*), which we summarize in section 1.2 and discuss in detail in [Kaelbling and Lozano-Pérez, 2011, 2012a]. 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 geometric models that support motion planning. In this paper, we extend the HPN approach to

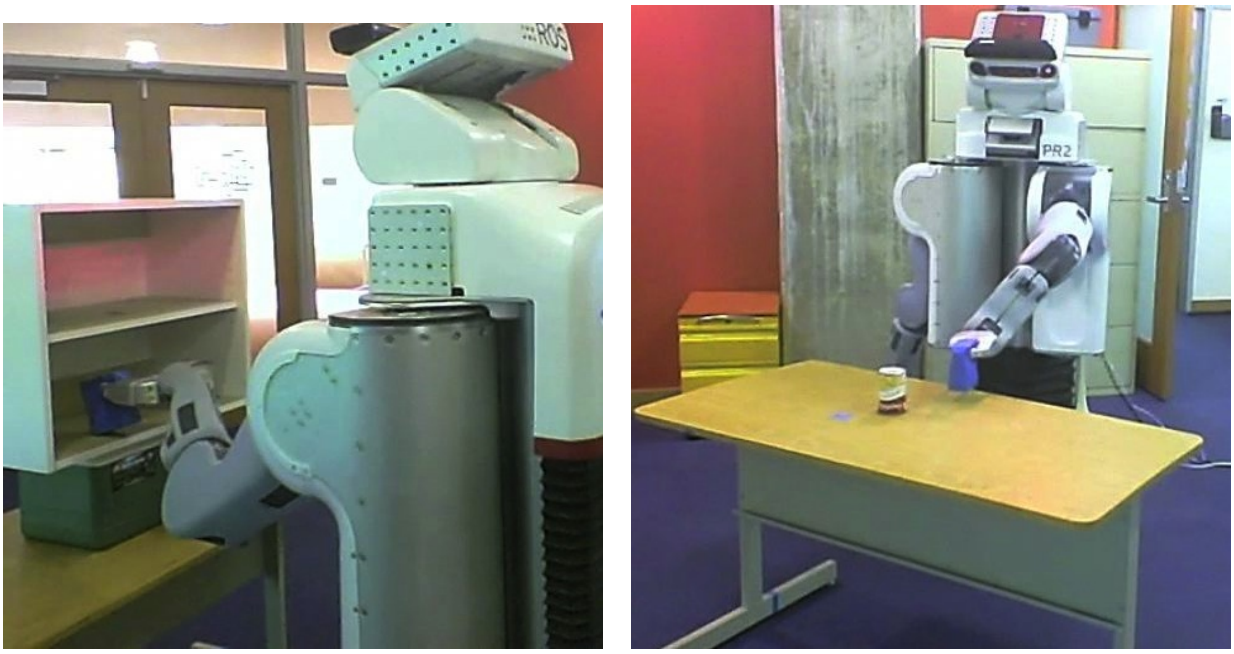


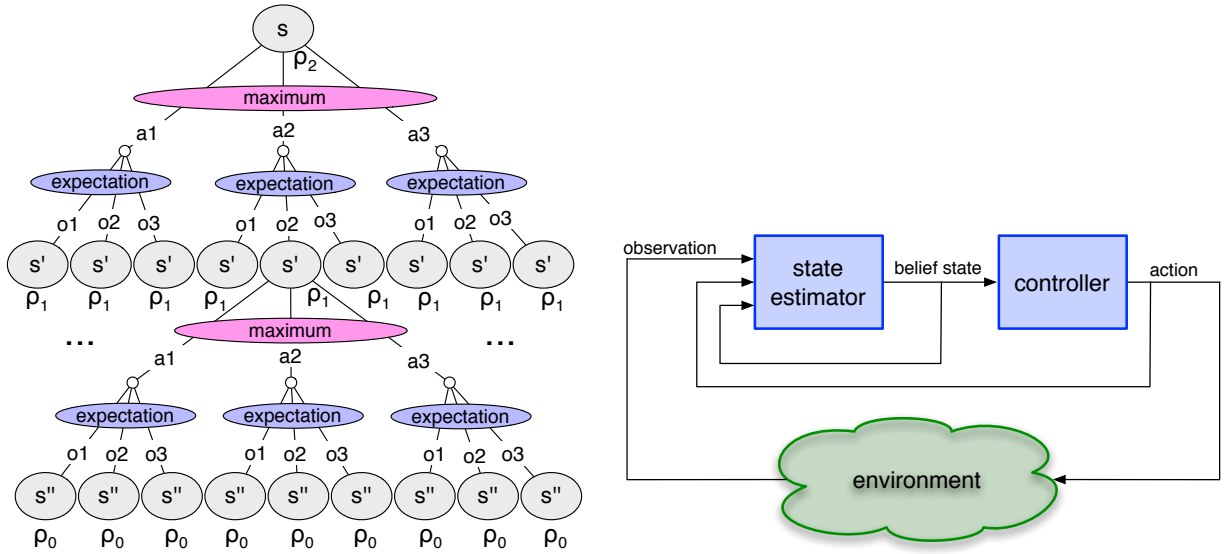
Figure 1: PR2 robot manipulating objects

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 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, such as disaster relief or managing logistical operations in a dynamic uncertain domain.

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(a) 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 value of a probabilistic node is computed as the expected value of its children, and the value of an



(a) Probabilistic policy tree

(b) POMDP estimation and control architecture

Figure 2: Models of decision-making in uncertain domains.

action-choice node is computed as the maximum of the values of its children. 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 address 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 2(b). 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 a policy that maps 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 that specify 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-conditions 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 HPN in observable domains

HPN [Kaelbling and Lozano-Pérez, 2011, 2012a] 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 simple conjunctions of a few logical requirements.

In a continuous space, pre-images may 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.

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 generated values are much more likely to be successful than ones chosen through an arbitrary sampling or discretization process.

To handle execution errors robustly, 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."

1.3 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., those of 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.

The work of Dogar and Srinivasa [2012] comes closest among existing systems to satisfying 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 intended to gather 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 nor do they reason hierarchically.

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; we review this related work in detail in [Kaelbling and Lozano-Pérez, 2012a]. 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 is the work of Levihn et al., which addresses 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.

There have been attempts to formulate and solve problems of planning robot motions under uncertainty in non-probabilistic frameworks dating back to the "pre-image backchaining" paper [Lozano-Pérez et al., 1984]. This work 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 are many formulations of 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].

The 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 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]. Recently, Bonet and Geffner [2011] have demonstrated that planning under partial observability in non-probabilistic domains can be, under some assumptions, reduced to classical replanning, resulting in an approach that is very efficient, when it applies.

Regression-based planning has been 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, using 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. However these systems do not use a probabilistic model of uncertainty or extend to continuous domains such as robot motion planning.

There is an enormous literature on hierarchical planning for deterministic domains; some of these ideas are also present in work on reinforcement learning (Barto and Mahadevan [2003] present a good survey) and planning under uncertainty (Mausam and Kolobov [2012] discuss this literature). Our approach to hierarchy is applied to determinized models, and so it is more closely related to the deterministic hierarchical planning literature (the relationship is outlined by Kaelbling and Lozano-Pérez [2012a]).

2 Example problem domain

The methods described in this paper provide a general-purpose approach to planning in complex uncertain domains, which encompasses a wide variety of problems inside and outside robotics. We go on to use the approach to formulate a method for planning and execution of general-purposes pick-and-place tasks, and further specialize that to work on a Willow Garage PR2 robot. In this section, we introduce the domain in detail. We have performed experiments in a simulated version of this domain, with noise in both perception and action. In addition, some examples have been executed on the real robot and are illustrated in section 6 and in the multimedia extensions (see Appendix A).

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, but much less error in the arm positioning relative to the base, and 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.

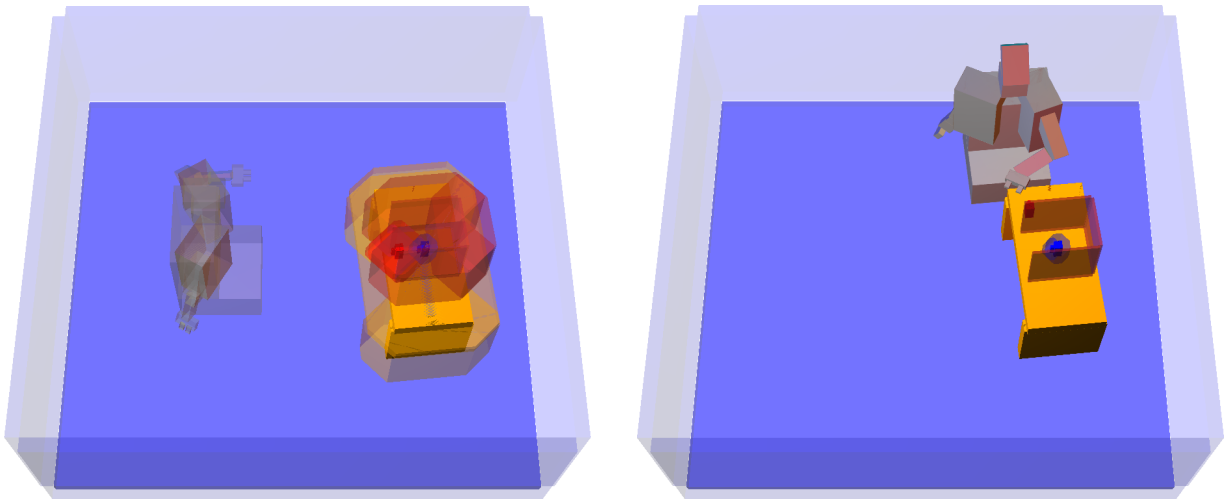
We assume that all of the objects are unions of convex polyhedra that are extrusions along the z axis (that is, they have a constant horizontal cross-section) and that their shapes are known in advance. The world state is not dynamic, in the sense that objects are always at rest unless they are in the 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 of 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 point cluster. 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 increase robustness in grasping objects, we employ a simple reactive grasping procedure [Romano et al., 2011] that uses information from touch sensors on the fingers to recover from small errors in an object's location estimate.

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 must contain enough information to



(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 3: Example starting and ending belief states.

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 must also 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 not necessarily all) objects in the domain, and knows what their shapes are; it may have a very highly uncertain estimate of their poses. Figure 3(a) shows an initial belief state, in terms of 0.05-shadows of the objects. Shadows are defined in detail in section 5.2.1: 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 and a u-shaped 'cupboard'. There is substantial uncertainty about their poses; their shadows are drawn to indicate the geometric extent of the uncertainty.

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

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 the possibility of false negative observations is not currently addressed.

We handle pose estimation 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 of multiple tangent spaces together with regular real spaces for the other dimensions can be taken, 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). We have integrated the approach of Wong et al. [2012], solving a constrained optimization problem to find the maximum likelihood arrangement of the objects that satisfies these constraints, and use that constrained maximum-likelihood estimate for motion planning.

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

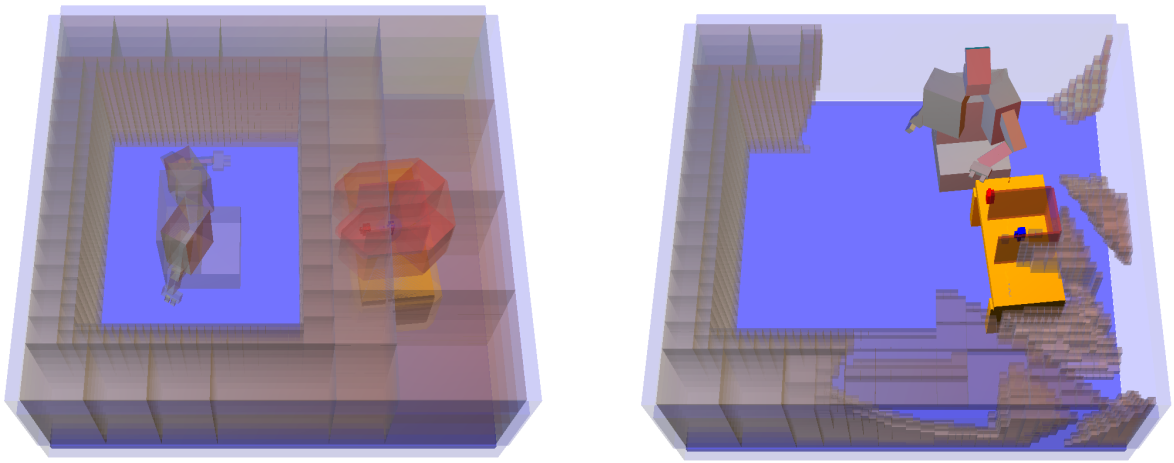
Figure 3(b) shows the terminal belief state from an execution trace that started with the belief state in figure 3(a) and had the goal of believing that the front cup was placed within a region of space on the left side of the table. Although it is somewhat occluded by the cupboard in this view, we can see that the one 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 other 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 maintain a representation of 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], which was 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. 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



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

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

Figure 4: Example starting and ending belief states including the observed space oct-tree.

between free and occupied regions of the environment; in our approach, those boundaries are well 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 4 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 4(a)), the robot knows the contents of the region of space nearby. 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 4(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 uncertain domains

To apply HPN in real-world robotics domains, we must extend it to address uncertainty both about the outcomes of actions and about the current state of the world.

3.1 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 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.

3.1.1 Determinizing the model

There are several 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 depends 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 logical 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 with value discounting. In fact, finite-horizon and infinite-horizon discounted MDPs are both subclasses of stochastic shortest path problems (SSPPs) [Barry et al., 2011, Bertsekas and Tsitsiklis, 1996, Mausam and Kolobov, 2012], and so we focus on SSPPs.

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.*

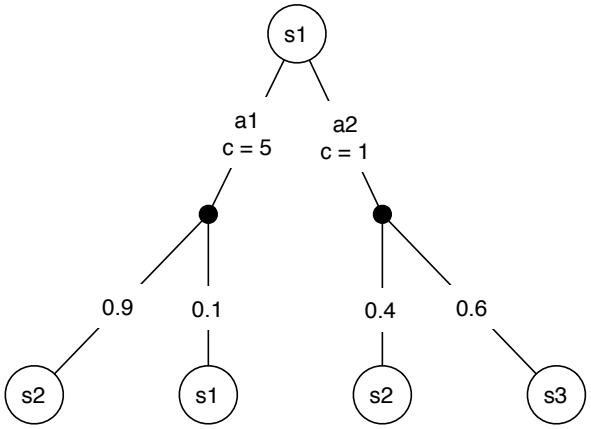
Definition 2. *A* stochastic shortest-path problem *(*SSPP*) $\langle S, A, T, C, G\rangle$ is an* MDP *in which all rewards are negative, there is a set $G \subset S$ of goal states, and the objective is to minimize the total expected cost (negative reward) incurred before reaching a state in G and terminating. We define cost function $C(s, a)$ in the* SSPP *to be $-R(s, a)$ in the original* MDP*, so that all costs are strictly positive.* [1]

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. We will convert an SSPP into a *determinized* SSPP as follows.

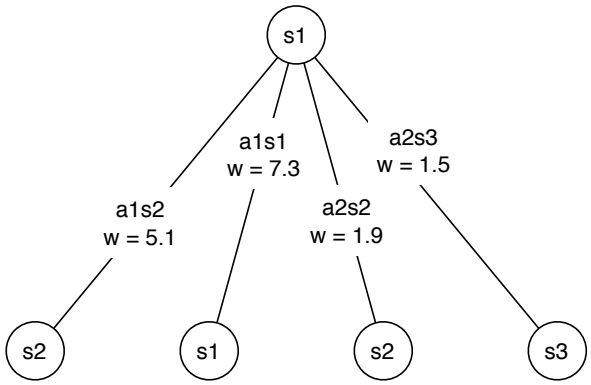
Definition 3. *A* determinized SSPP *is a tuple $\langle S, A', W, G\rangle$ where: $\langle S, A, T, C, G\rangle$ is a* SSPP*; S is a set of states which are nodes in a graph; A' is a set of actions (a, s'), so that $(s, a, s') \in S \times A \times S$ is a directed arc from node s to node s'; and W 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. When $W(s, a, s') = C(s, a)$, this is referred to as the *all-outcome determinization* [Mausam and Kolobov, 2012], because any possible action can be selected, at the same cost. We will define another weight function that depends on the likelihood that each transition will occur.

[1] This definition matches the "weak" definition of a stochastic shortest-path MDP in Mausam and Kolobov [2012]; their "strong" defines a somewhat larger class of problems, allowing costs to be negative, but imposing an additional global constraint; the global constraint is more difficult to verify, so we will use the simpler "weak" definition.



(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 5: Probabilistic search tree and its deterministic approximation.

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. See [Blum and Langford, 1999]. □

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. Keyder and Geffner [2008] and Barry et al. [2011] offer one strategy, in which it can be more difficult to make trade-offs between cost and likelihood of success; we use a parameterized combination of the two objectives, defined below.

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

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

Figure 5(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 5(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 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.

3.1.2 Regression with costs

In the HPN framework, we 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 search through

the space of *subsets* of the state space. We will use logical expressions to compactly denote large or infinite subsets of the state space, so the complexity of this search approach will be independent of the size of the underlying state space, but dependent on the sizes of the logical expressions. 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. *A* regression cost problem *(RCP) is a tuple $\langle N, A, W' \rangle$ derived from a* DSSPP *$\langle S, A, W, G \rangle$: 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:*

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

The set N of pre-images is defined recursively starting from the goal set G of the original DSSPP*:*[2]

- *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.

3.1.3 Markov HPN

We extend the basic HPN algorithm [Kaelbling and Lozano-Pérez, 2012a], to apply to domains with outcome uncertainty by retaining the same depth-first planning and execution architecture, but additionally monitoring the effects of actions 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 obviously 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, γ, α, world):
    p = PLAN(s_now, γ, α)
    while s_now ∈ envelope(p)
        i = arg max_i s_now ∈ g_i(p)
        if ISPRIM(ω_i(p))
            s_now = world.EXECUTE(ω_i(p))
        else
            s_now = MHPN(s_now, g_i(p), NEXTLEVEL(α, ω_i(p)), world)
    return s_now
```

[2]Note that this set is potentially infinite and it is ever explicitly constructed; it is partially realized dynamically during the search process.

MHPNTOP($s_{now}, \gamma, world$):

while $s_{now} \notin \gamma$

s_{now} = MHPN($s_{now}, \gamma, \alpha_0, world$)

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 will write $g_i(p)$ to stand for the ith operator, ω_i, in plan p.

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 has been proven 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 pre-image 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 be reached. This is 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 logical assertions and it only needs to test whether the current, fully specified, state satisfies these assertions.

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.

Figure 6 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. As described in detail in [Kaelbling and Lozano-Pérez, 2012a], we create abstract versions of operators by postponing consideration of some of the preconditions. Assume that we have an abstract version of *Op1* that has postponed the precondition A. Given the goal of achieving a state in which $E\&D$ is true, we first 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

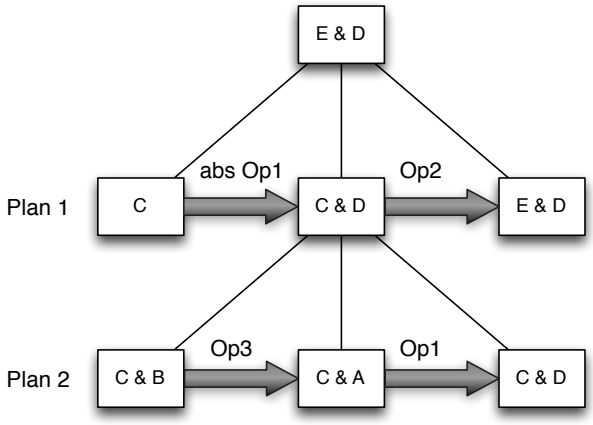


Figure 6: MHPN execution example

true pre-image, which is C, and executes the associated operation. In this case, it is an abstract version of *Op1*, so HPN is called recursively with $\gamma = C \ \& \ D$ with the 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 several different possible outcomes.

1. C remains true and A becomes true, in which case we would go on to execute the primitive *Op1*. This is the "nominal" outcome that was intended by the planner.

2. 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*.

3. C remains true, A does not become true, but B remains true, in which case we would execute the primitive *Op3* again.

4. C remains true, A does not become true, and B 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. It would attempt to replace plan 2 with a new plan to achieve $C \ \& \ D$ that does not depend on B, and begin executing it.

5. C becomes false. In this case, as in the previous one, 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 guarantee the global validity 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,

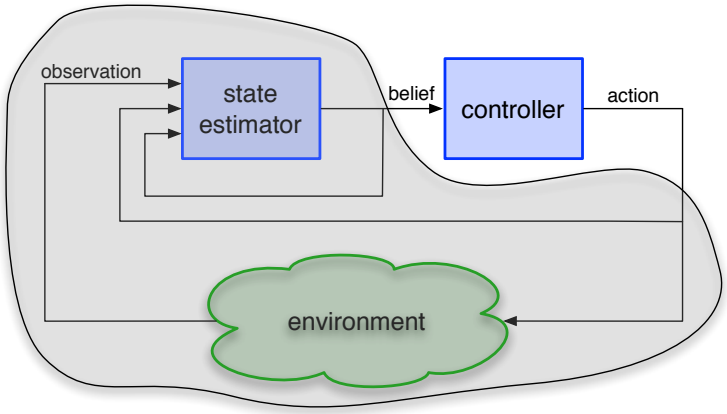


Figure 7: From the perspective of the controller in a POMDP, the belief-state estimation module becomes part of the external environment, and the job is to map belief states into actions in such a way as to drive the belief state into a desired set.

the recursion will return to a 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 some state in the goal set. For finite deterministic environments, Kaelbling and Lozano-Pérez [2012a] state 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 is reachable from the initial state. In the probabilistic case, as long as each plan that is constructed has a finite probability of success, then eventually a goal state will be reached.

3.2 Uncertain state

When there is uncertainty about the current state of the world, we can formalize the problem as a POMDP. Assuming a state estimator has been specified, the control problem can be seen as shown in figure 7. 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.

As a 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.

3.2.1 HPN in belief space

Planning in belief space is generally quite complex, because it requires 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 a very high-dimensional continuous space of configurations of a robot and many objects. We take direct advantage of this analogy and use logical predicates to specify properties of belief states, as our earlier HPN approach does for properties of geometric configurations. Pre-image backchaining allows the construction of high-level plans to achieve goals articulated in terms of those predicates; it will work effectively when the pre-images can also be described using relatively small sets of these predicates.

The problem of updating the state estimate based on an action and observation arises in two distinct guises in our approach: first, in the *state estimation* module as shown in figure 7 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 critical that the belief-state representation and update in the state estimation module be as accurate as possible: that belief state is our only proxy for the entire history of actions taken and observations made by the robot. However, the model used by the planning process can have considerably lower fidelity: errors made in planning due to model approximation can typically be corrected by observing a new resulting belief 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 algorithm to the BHPN algorithm, 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($b_{now}, \gamma, \alpha, world$):
  $p = \text{PLAN}(b_{now}, \gamma, \alpha)$
  while $b_{now} \in envelope(p)$
    $i = \arg\max_i b_{now} \in g_i(p)$
    if $\text{isPRIM}(\omega_i(p))$
      $obs = world.\text{EXECUTE}(\omega_i(p))$
      $b_{now} = b_{now}.\text{UPDATE}(\omega_i(p), obs)$
    else
      $b_{now} = \text{BHPN}(b_{now}, g_i(p), \text{NEXTLEVEL}(\alpha, \omega_i(p)), world)$
  return b_{now}

BHPNTOP($b_{now}, goal, world$):
  while $b_{now} \notin goal$
    $b_{now} = \text{BHPN}(b_{now}, goal, \alpha_0, world)$

After each primitive action (which may in fact involve calling a motion 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.

4 Logical characterization of beliefs for planning

We use a general-purpose planner that takes as input 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.

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, their pre-images, and the belief process dynamics.

This section describes a method for representing sets of belief states and their dynamics using symbolic operator descriptions, and shows how they can be used in regression-based planning and embedded into the BHPN recursive planning and execution architecture.

4.1 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 use this domain to illustrate the use of logical assertions to characterize beliefs, to show how operator descriptions can be used to specify approximate deterministic dynamics in stochastic domains, and to demonstrate the results of planning and execution.

4.1.1 Belief states

In general, in a discrete-state domain with n states, a belief state is a multinomial 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 8(a). We will use $\Pr_b(e)$ to denote the probability assigned to event e (a subset of the state space) by the distribution b.

4.1.2 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.

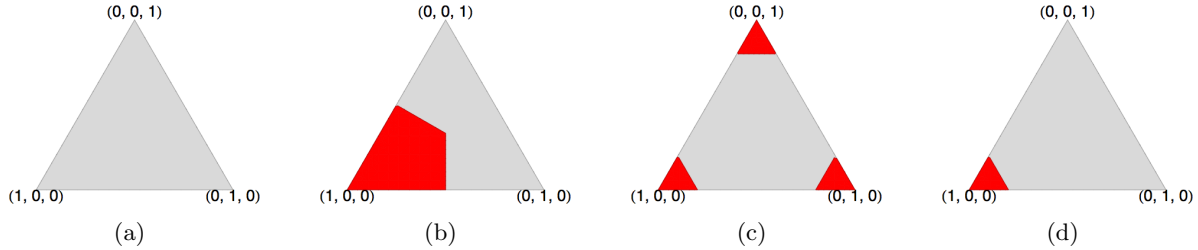


Figure 8: (a) Two-dimensional simplex represents the set of all possible belief states over a discrete state space with three states. (b) Subset of belief states in which state 0 is most likely. These belief states satisfy $MLLoc(l_0)$. (c) Subset of belief states in which one of the states is believed to be true with probability ≥ 0.8. These belief states satisfy $BVLoc(0.2)$. (d) Subset of belief states in which state 0 is believed to be true with probability ≥ 0.8. Note that this is the intersection of the previous two sets. These belief states satisfy $BLoc(l_0, 0.2)$.

Logical statements are made up of conjunctions of fluents. A *fluent* is a condition that can change over time, typically expressed as a logical predicate applied to a list of arguments, which may be constants or variables. 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, $\tau_f : args, b \rightarrow \{true, false\}$, where $args$ are the arguments of fluent f, b is a belief state, and the fluent $f(args)$ holds in belief state b if and only if $\tau_f(args, b) = true$. 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 by specifying their associated tests. 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:

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

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

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

- $BVLoc(\epsilon)$: *the value of the location of the object is believed* with probability greater than $1-\epsilon$:

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

  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 sets of belief states, in the three-location domain, corresponding to each of these fluents are shown in figures 8(b), 8(c), and 8(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; these are provided in [Kaelbling and Lozano-Pérez, 2012b].

4.1.3 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 whether the object is there.

Move When the robot attempts to move an object from location l_i to location l_j, 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 begin 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:

$$\Pr_{b'}(S = l_j) = b_i(1 - p_{fail}) + b_j \quad ,$$

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:

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

The variables l_{start} and l_{target} denote the initial and final locations of the object. The **effect** clause specifies fluents that will result from performing this operation, if the conditions in the **pre** clause are true before it is executed; the **choose** clause means that there is an instance of this operator schema for each possible binding of the choice variable; the **prim** is the primitive action associated with the operation and the cost is specified in the **cost** clause.

We let *Locations* indicate the universe of possible object locations in the domain.[3] Because the result is deterministic, the cost is just the base cost of executing the action[4].

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. Derivations supporting propositions made in the body of this paper appear in [Kaelbling and Lozano-Pérez, 2012b].

[3] In general, the domains of variables may be infinite.

[4] In this paper, all such costs are 1; it remains for us to develop a more general treatment of costs in hierarchical problems.

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 takes the action (A) of looking in a location l_i, it may either make an observation $O = true$ or $O = false$, depending on the state (S) with these probabilities:

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

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, 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 for the purposes of planning: 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 and not to have goals that require its dispersion, so we only go on to formalize the effects of cases 1 and 4.

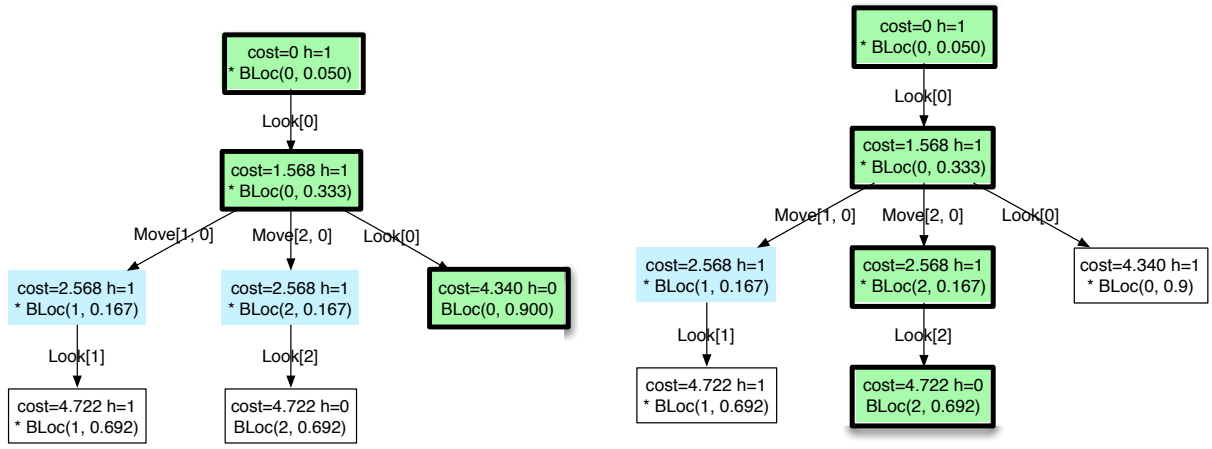
The effects on $\Pr(S = l_{target})$ when the action is to look at location l_{target} and the object is detected are characterized by this operator description:

$\textsc{LookToVerify}(l_{target})$:
effect: $BLoc(l_{target}, \epsilon)$
pre: $BLoc(l_{target}, lookPosRegress(\epsilon))$
prim: $\textsc{LookPrimitive}(l_{target})$
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. The *lookPosRegress* function takes a value ϵ and returns a value for ϵ' such that, if the object is believed with probability at least $1 - \epsilon'$ to be in the target location before a look operation is executed and the object is detected, then it will be believed to be in that location with probability at least $1 - \epsilon$ afterwards.

Proposition 3.

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



(a) Plan 1 (b) Plan 2

Figure 9: 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.

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. In [Kaelbling and Lozano-Pérez, 2012b] we also define an operator that characterizes the effect of looking at a location l_{target} in order to rule it out (case 4).

4.1.4 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 should 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 9(a) shows the search tree for the first planning problem. Nodes with bold outlines represent the solution path; other shaded nodes 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—this heuristic is neither admissible nor particularly tight in general). 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.

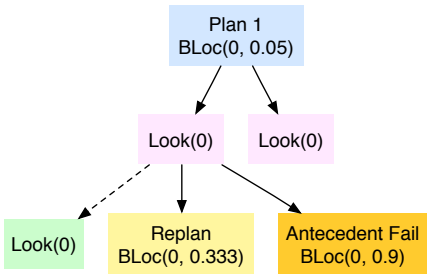
There is only one way to achieve believing the object is in location 0 with high confidence, which is to look in location 0 to verify that the object is there. The move operation has sufficient error associated with it, that after a move, it will never be possible for the object's location to be known 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 consider 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. Continuing the search, we find that at least one look operation is necessary before attempting to move the object, in order to establish that the object is in the starting location of the move. The plan to move the object is, thus, more costly than the (over) optimistic plan of looking twice in location 0 to verify that the object is there, and so we adopt the plan of looking in location 0. Note that by changing α in the determinization cost formula, we can change the behavior of the system so that it would prefer the plan of moving the object, because it is more likely to succeed.

Figures 10 and 11 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 titled `Plan i`, plan steps are below plan nodes connected by solid arcs, primitive actions are leaves connected by arcs, and indications of replanning steps are labeled as `Replan` or `Antecedent Fail`. In particular, when a plan step does not have the expected outcome, a 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 a 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.

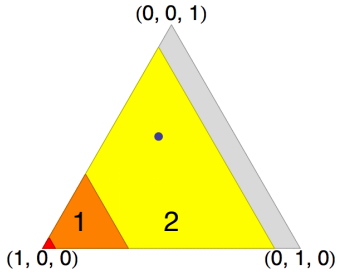
Following the tree is the pre-image sequence of the corresponding plan, drawn in the belief simplex. The small region at the lower left vertex (red) is the goal, the region labeled 1 is the pre-image of the goal under the planned action, the region labeled 2 is the pre-image of region 1, and the region labeled 3 is the pre-image of region 2. The unlabeled 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 9(a), shown as the first two look nodes in the figure 10(a), and as the colored pre-image sequence in figures 10(b) and 10(c). 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 10(c), it has gone outside the envelope of the plan, so replanning is triggered.

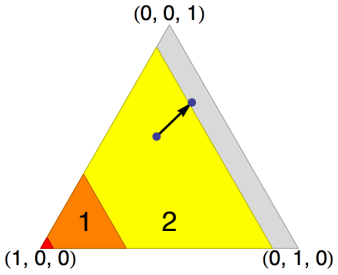
- Plan 2 (corresponding to the path in figure 10(e)) is shown in figure 10(d). It has as its first step to look in location 2, but when that step is executed, the object is not observed to be



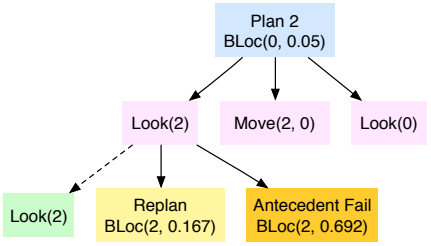
(a) Plan 1: Look(0), Look(0)



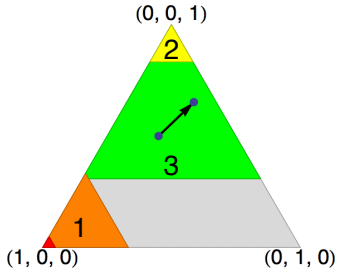
(b) Plan 1: Look(0), Look(0)



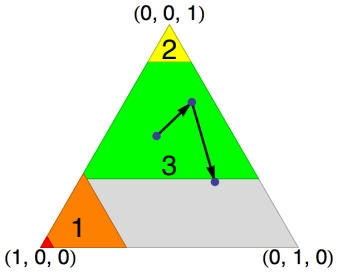
(c) Look does not see object in 0; belief state exits envelope of plan; replanning is triggered.



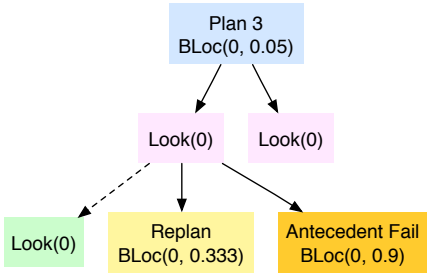
(d) Plan 2: Look(2), Move(2, 0), Look(0)



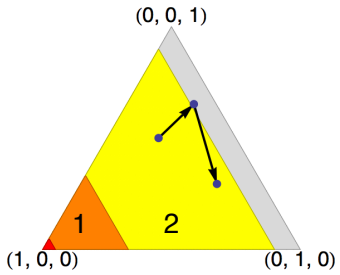
(e) Plan 2: Look(2), Move(2, 0), Look(0)



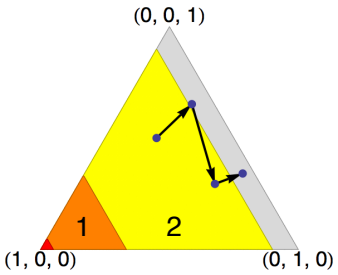
(f) Look does not see object in 2; belief state exits envelope of plan; replanning is triggered.



(g) Plan 3: Look(0), Look(0)

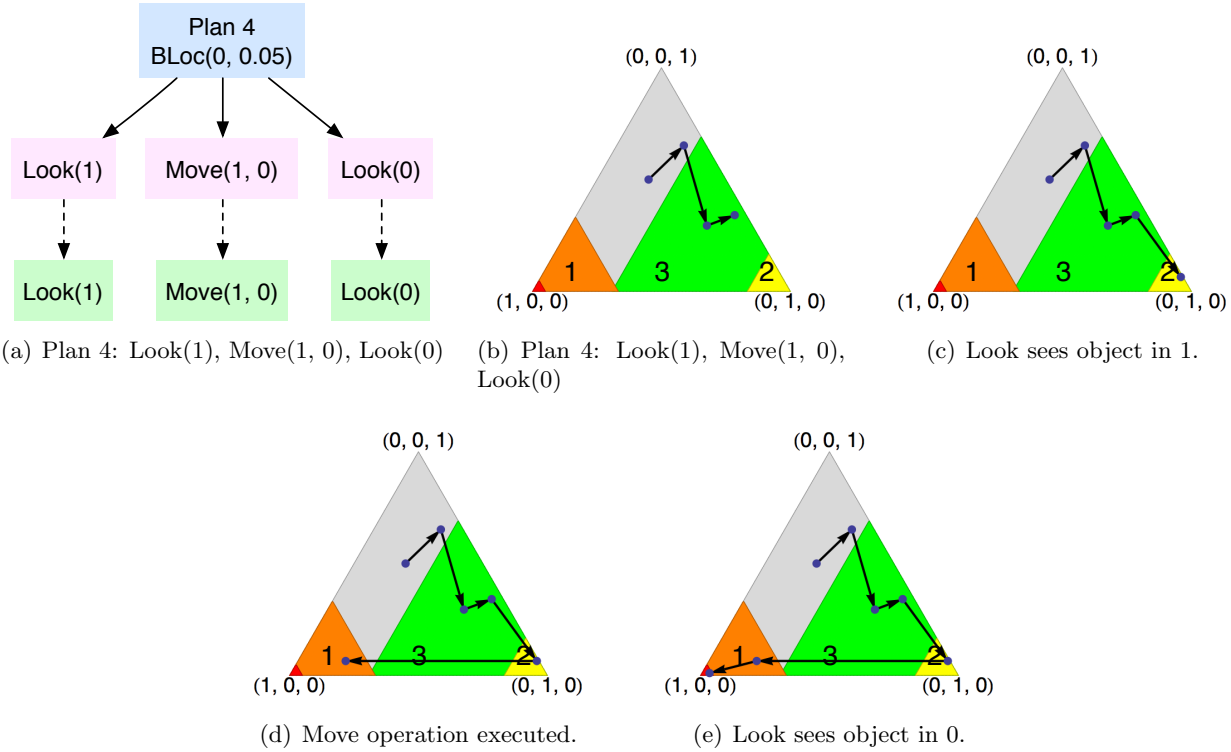


(h) Plan 3: Look(0), Look(0)



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

Figure 10: First three plans and execution steps in the three-location domain. Each row corresponds to a plan and its execution in the domain. Figure 10(b) shows the initial belief state as a dot, the goal set as a very small triangle in the bottom-left corner, and the pre-images of two plan steps are labeled 1 and 2. Figure 10(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 11: The final plan and execution steps in the three-location domain.

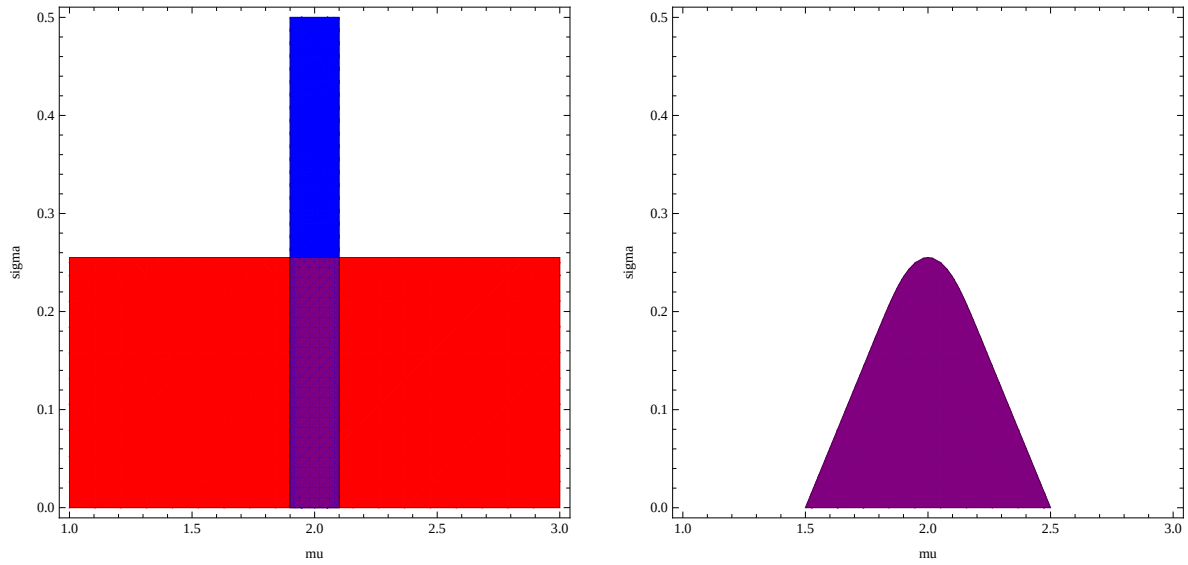
in location 2, so the belief state again exits the envelope of the plan (shown in figure 10(f)), and replanning is triggered.

- Plan 3, shown in figures 10(g) and 10(h), again looks in location 0 because it has again become more likely, but after execution, the robot does not see the object and replanning is triggered (figure 11(b)).
- Plan 4, shown in figures 11(a) and 11(b) looks in location 1 and sees the object (figure 11(c)), moves the object to location 1 (figure 11(d)), and finally looks in location 1 and sees the object (figure 11(d)).

This example demonstrates that a small set of discrete fluents can be used for effective planning in belief space.

4.2 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 that the variance be below some threshold. Generally, we would like to derive requirements on beliefs from requirements for action in the physical world. For example, 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



(a) Regions of a Gaussian belief space captured by fluents: $BV(X, 0.05, 0.5)$ is low wide region; $ModeNear(X, 2, 0.1)$ is tall thin region

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

Figure 12: Fluents in continuous belief space

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 1m. 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[5] with probability at least $1 - \epsilon$:

$$\tau_{BV}((X, \epsilon, \delta), b) := \Pr_b(|X - \overline{X}| < \delta) \geq 1 - \epsilon \ .$$

- $ModeNear(X, v, \delta)$: the mode of random variable X is within δ of value v:

$$\tau_{ModeNear}((X, v, \delta), b) := |v - \overline{X}| < \delta \ .$$

- $B(X, v, \epsilon, \delta)$: the value of random variable X is within δ of value v with probability at least $1 - \epsilon$:

$$\tau_B((X, v, \epsilon, \delta), b) := \Pr_b(|X - v| < \delta) \geq 1 - \epsilon \ .$$

[5]We will use the notation $\overline{X}$ to mean the mode of random variable X throughout the paper.

4.3 Pre-images

There are several ways in which taking an action can affect these fluents: information can be gained through observation; information can be lost through actions with probabilistic effects; and 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 sections.

4.3.1 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 actions a and observations 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 cannot be computed analytically, but it can be estimated 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:[6]

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

Perhaps 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 observations, which is not true in general (as we already saw for discrete observations in the previous section). Additional preconditions can be added, for example if there are visibility 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 mass within δ of the mode is

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

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

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

The *obsRegress* function takes a desired ϵ and δ, and the standard deviation σ_o of an observation, and returns an ϵ' such that if there is at least $1 - \epsilon'$ probability of being within δ of the mode

[6]The observation noise in real robots is seldom actually independent and identically distributed (IID) yet this model of information gain based on successive observations is sufficiently accurate for planning.

before the observation, then there will be $1-\epsilon$ probability of being within δ of the mode after the observation.

Proposition 4.

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

In a Gaussian, the condition $BV(X, 0.05, 0.5)$ corresponds to a requirement that σ be less than 0.255. The regression of this condition under an observation with $\sigma_o = 0.4$ is that σ be less than 0.331. 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: $BV(X, changeRegress(\epsilon, \delta, \sigma_u), \delta)$
cost: 1

Although the results of changing X may be stochastic, the belief-state update is deterministic. The *changeRegress* function takes a desired ϵ and δ and the standard deviation of the effects of an action σ_Y, and returns an ϵ' such that if there is at least $1-\epsilon'$ probability of being within δ of the mode before the transition, then there will be $1-\epsilon$ probability of being within δ of the mode after the transition.

Proposition 5.

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

If $\epsilon < 1 - \text{erf}\left(\frac{\delta}{\sqrt{2}\sigma_Y}\right)$, then there is no degree of prior certainty that will guarantee that, after the action, the BV condition will be satisfied. The regression of the condition $BV(X, 0.05, 0.5)$ under a transition with $\sigma_Y = 0.2$ is the condition that σ be less than 0.158. We can see that only a smaller variance is allowable if the next action is a transition, because the motion will increase the variance.

4.3.2 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, δ)
pre: *ModeNear*$(X, v - u, \delta)$
cost: 1

However, in general, any observation other than the most likely one 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 cause a belief update that violates a *ModeNear*(v, δ) condition depends on the uncertainty in the distribution at the time of the observation characterized as PNM$(X, \delta_b) > 1 - \epsilon$, as well as v and δ.

In the planning process, if an *Observe* action is being taken to achieve fluent $BV(\epsilon, \delta_b)$ and there is a *ModeNear*(v, δ) fluent in the goal at the same time, then the probability that the *ModeNear* fluent will be violated is *probModeMoved*$(\epsilon, \delta_b, v, \delta)$.

Proposition 6.

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

where

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

So, we can rewrite the OBSERVE operator as:

OBSERVE(X):

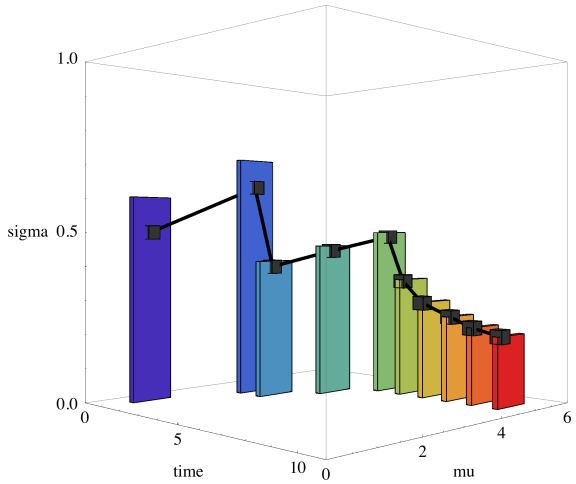
effect: *ModeNear*(X, v, δ), $BV(X, \epsilon, \delta_b)$
pre: *ModeNear*(X, v, δ), $BV(X, obsRegress(\epsilon, \delta_b, \sigma_o), \delta_b)$
cost: $1 - \log probModeMoved(\epsilon, \delta_b, v, \delta)$

4.4 Example execution

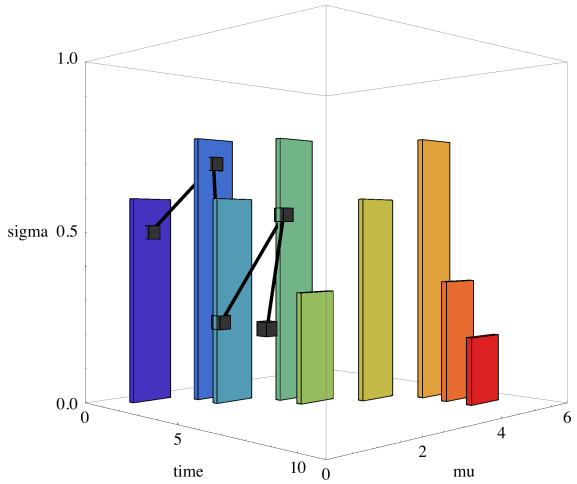
To provide intuition about planning in belief space with a Gaussian distribution on a single underlying continuous variable, we show two example planning and execution runs. We can think of this as a simple problem of navigation in one dimension, in which the robot can move by a continuous offset from 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, 2012a], 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.

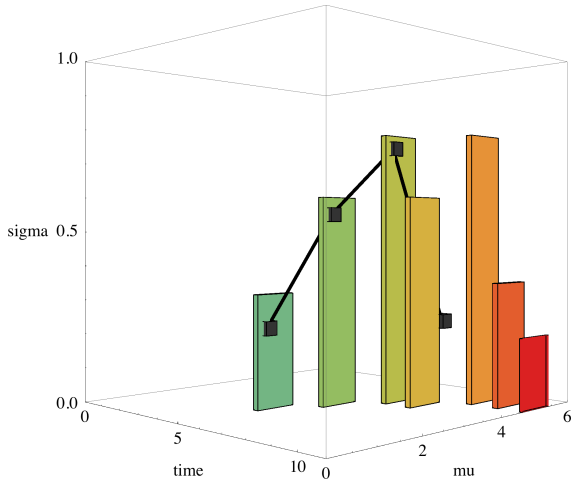
To limit the branching factor, the choice statement in the MOVE operator considers actions that would move the mode by +1, -1, or by the total distance from the mode of the current belief distribution to the goal mode.



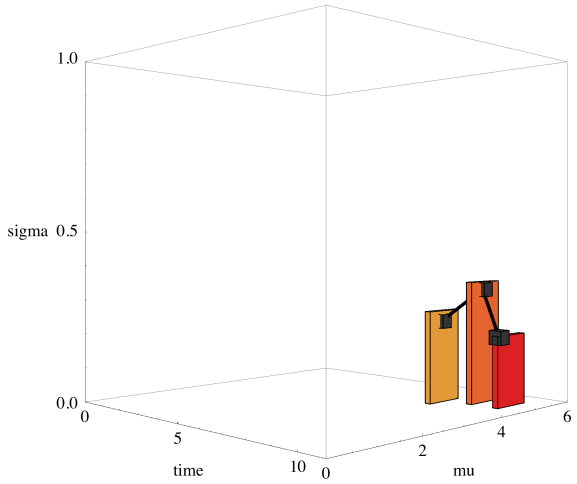
(a) Belief space plan, shown in colored bars and execution trajectory shown in black. The smallest bar in the right front is the goal in μ, σ space; the pre-images are shown going back through time.



(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 13(b), in which execution trajectory deviates again.



(d) Final plan segment.

Figure 13: Example plans and execution trajectories in the one-dimensional continuous domain.

MOVE$(t, u, \delta, b_{now}, \gamma)$:

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

The standard deviation of the transition distribution for action u, $\sigma_u = \alpha|u|$, is proportional to the magnitude of the control. We have to add a declaration to the planner that the MOVE operator affects BV fluents; during regression, if $BV(\epsilon, \delta)$ is in the goal, and a MOVE operator is applied, we must replace $BV(\epsilon, \delta)$ with $BV(changeRegress(\epsilon, \delta, \alpha|u|), \delta)$.

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.

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 13(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), LOOK, MOVE(1), MOVE(1), followed by 5 LOOKs. The axes of the graph are the μ and σ of the belief and the time step of the process. The 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 13(b) through 13(d) shows a run with $\sigma_o = 0.25$ and $\alpha = 0.5$. In this case, fewer observations are necessary, but the planner elects to take only single-step actions in order to have the opportunity to make observations that will keep the uncertainty down below the level at which observation actions are no longer applicable.

In figure 13(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 13(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 13(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 of belief space, and use execution monitoring and replanning to handle unexpected outcomes of those plans, eventually achieving a stated belief-space goal.

5 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.

5.1 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 $\mathcal{S}_{obs}$. The primitives are always executable because the preconditions guarantee the existence of a feasible solution before the primitive is allowed to be executed. So, for example, the running time for the RRT does not have to be limited *a priori*; the RRT can be re-started multiple times until a path is found.

- PICKPRIMITIVE(*obj*, *grasp*) causes the robot to 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).
- 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, as well as a 3D point cloud which is used to track which parts of the space have been observed.

5.2 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.

5.2.1 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 relevant information. It is critical to characterize the robot's knowledge of 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 $[-\pi, +\pi]$.

This distribution is non-Gaussian and cannot be represented exactly. Instead, we use a sigma-point approximation [Julier and Uhlmann, 2004]. Define the α-sigma points of multivariate Gaussian distribution μ, Σ as follows:

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

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,

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

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

$$D = \{f(c) \mid c \in \text{SIGMAPOINTS}(\overline{PQ}, \Sigma_J, 1)\} ,$$

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, $S(o_1, o_2, \epsilon, b) = \textsc{Shadow}(o_1, \mu_{\pi(o_1,b)-\pi(o_2,b)}, \Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \epsilon)$ as follows:

$$\textsc{Shadow}(o, \mu, \Sigma, \epsilon) = \bigcup_{j=1}^{numParts(o)} \textsc{convexHull}(\{vol(o_j, \pi) \mid \pi \in \textsc{sigmaPoints}(\mu, \Sigma, \sqrt{\chi^2_4(1-\epsilon)})\})$$

Note that $\sqrt{\chi^2_4(1-\epsilon)}$ 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. This is used to determine a 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; the shadow of the entire object is the union, over each of these parts, of the convex hull of the placement of the part at each of the sigma points (which are poses) relative to its centroid.

The ϵ-shadow is constructed with the objective that, with probability $1-\epsilon$, it contain the volume taken up by o if it is placed at a pose 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, and determine the necessary pre-images under observation and transition actions, it would be necessary to formulate expressions for PNM and its pre-images in terms of the 4D joint distribution. For expedience in our implementation, we define a vector-valued $\textsc{pnm}_\pi$ 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

$$\textsc{pnm}_\pi(\Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \delta) \geq 1 - \epsilon$$

will be true if the PNM condition is satisfied componentwise.

5.2.2 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 poses of the objects and/or the robot. The approach could be 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:

$$\tau_{BVRelPose}((o_1, o_2, \epsilon, \delta), b) := \textsc{pnm}_\pi(\Sigma_{\pi(o_1,b)-\pi(o_2,b)}, \delta) \geq 1 - \epsilon \ .$$

- *PoseModeNear*(o, π, δ) : 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 component-wise absolute differences (with the angular difference appropriately wrapped), and $\leq$ holds for the vector if it holds for all four components individually.

- *ConfModeNear*(c, δ) : true if the mode of the distribution of the 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, ϵ) : 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 r is contained in r:

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

- *BContents*(r, x, ϵ): true if the contents of region r are known with certainty greater than $1-\epsilon$.

  This 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 entire region has been observed:

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

- *BClearX*(r, x, ϵ) : 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, whose ϵ shadow did not overlap the region, then with probability at least $1 - \epsilon$ the region would be 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 $1 - \epsilon$:

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

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 [Kaelbling and Lozano-Pérez, 2012b].

5.3 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.

5.3.1 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.

PICK(o, g, b_{now}, γ):

effect: $BHolding(o, g, \epsilon)$

pre: $o \neq None$ **[0]**

$BVRelPose(o, \text{ROBOT}, \epsilon_{pickPlan}, \delta_{pickPlan})$ **[1]**

choose: $\pi \in \{\pi(o, b_{now}), \text{GENERATEPARKING}(o, g, b_{now}, \gamma)\}$

choose: $motion \in \text{GENERATEPICKPATH}(o, g, \pi, b_{now}, \gamma)$

$PoseModeNear(o, \pi, \delta_{pickPlan})$ **[2]**

$BClearX(sweptVolume(motion), \{o\}, \epsilon_{clear})$ **[2]**

$BHolding(None, None, \epsilon_{holding})$ **[3]**

$ConfModeNear(pregraspConfig(motion), \delta_{grasp})$ **[3]**

$BVRelPose(\text{ROBOT}, o, \epsilon, \delta_{grasp})$ **[4]**

sideEffects:

$ConfModeNear(C, D) = None$ **[0, 1]**

prim: PICKPRIMITIVE(o, g)

The main section of the operator description is a list of preconditions; each precondition is followed by a number indicating at what abstraction value of hierarchical planning that precondition is being considered.[7] At the most abstract value (0) the only requirement is that o not be *None*, which states 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, selected from a region that is out of the way of the current pick-and-place problem. 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 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 an arbitrary home configuration to the pre-grasp configuration. Paths are always planned (by a visibility-graph planner[Lozano-Pérez and Wesley, 1979]) 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 at 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 an immovable obstacle. It prefers to generate paths that avoid movable obstacles, but will plan to move 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).

[7] Abstraction levels are discussed in detail in [Kaelbling and Lozano-Pérez, 2012a].

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 π; 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 will fail and be 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 to make it likely that the primitive grasp action will succeed.

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}, $\epsilon_{\mathit{clear}}$, $\epsilon_{\mathit{holding}}$, $\epsilon_{\mathit{pickPlan}}$, and $\delta_{\mathit{pickPlan}}$. 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.

5.3.2 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 with respect to the global coordinate frame as the object is observed and the belief state is updated. Note that we require detailed knowledge of the object's current pose before making a detailed plan for placing it; this enables choice of a grasp to depend on the obstacles at both the starting and destination poses.

PLACE$(o, \pi, \mathit{region}, \epsilon, b_{now}, \gamma)$:

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

5.3.3 Move robot

This operation moves the mode of the belief distribution on the robot's configuration to a desired value. It is impossible to guarantee a tolerance δ tighter than the tolerance of the underlying controller.

MOVEROBOT$(c, \delta, b_{now}, \gamma)$:

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

When the robot moves, our information about the relative poses of the robot and other objects changes. If there is a fluent $\mathit{BVRelPose}(o_1, o_2, \epsilon, \delta)$ in the goal and either $o_1 =$ ROBOT or $o_2 =$ ROBOT, then in the pre-image of the goal under a motion action we must replace it with

$$\mathit{BVRelPose}(o_1, o_2, \mathit{changeRegress}(\epsilon, \delta, \sigma_u), \delta) \ .$$

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 4.3.1 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 this is not a legal move.

5.3.4 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.

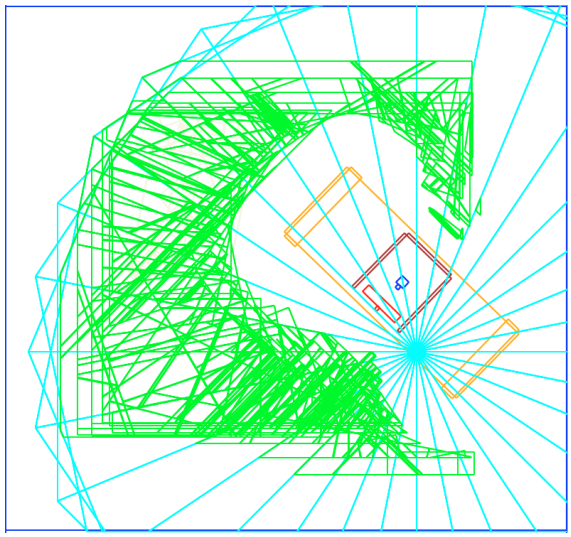
To come to know the pose of one object relative to another, there are essentially two methods: to observe them both at the same time, or to observe them sequentially without inducing too much error in the relationship between the robot's poses at those observations. Here, we take the simple, but weak approach of reducing the requirement that we simultaneously 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. The following operator description characterizes conditions for coming to know the pose of an object with respect to the robot.

LOOKOBJ(o, ROBOT, $\epsilon, \delta, b_{now}, \gamma$):
effect: $BVRelPose(o, \text{ROBOT}, \epsilon, \delta)$
pre: $\epsilon > 0$ **[0]**
choose: $(motion, viewCone) \in$ GENERATELOOKMOTION(o, b_{now}, γ)
pre: $BClearX(viewCone, [o], \epsilon_{huge})$ **[1]**
$ConfModeNear(viewConf(motion), \delta_{\text{LOOK}})$ **[1]**
$BVRelPose(o, \text{ROBOT}, obsRegress(\epsilon, \delta, \sigma_o), \delta)$ **[2]**
prim: LOOKATOBJPRIMITIVE(o)

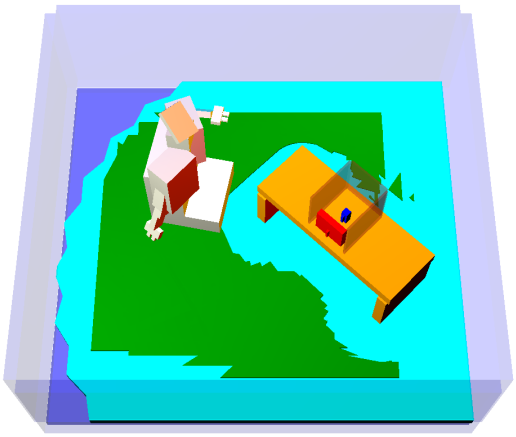
The generator for LOOK motions finds collision-free configurations for the robot so that it has an unoccluded view of the object. The first step is to characterize (x, y, θ) configurations of the robot base so that the front of the robot points toward the object, within a specified max angle ϕ, which is a conservative estimate of the robot's field of view ($\phi = \pi/4$ in our experiments). For a given orientation θ of the robot, the (x, y) base positions with this property define circle sectors (with central angle ϕ) on either side of the line from the object to the base center. We approximate these "slices" with isosceles triangles with their apex at the object and with side length determined by the sensor range. Figure 14(a) shows a large number of these triangles calculated for uniformly sampled base θ values. Each of these triangles represent a set of base poses from where the robot could potentially have the object in its field of view. However, there is no guarantee yet that these configurations are collision free.

For each sampled θ, we compute the set of (x, y) configuration space obstacles for the base[Lozano-Pérez, 1983] relative to the objects in the environment. These represent positions of the base that give rise to collisions. We then cut out these obstacles from the computed θ slices. The resulting free space representation is a set of overlapping polygons in figure 14(a). These polygons represent configurations for the base such that the target object 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 14(b) it cannot actually see the target location because it is occluded by the cupboard and another obstacle as seen in figure 14(c). The generator therefore samples locations in the valid free space regions and does a detailed test for visibility (by coarse ray tracing) to find candidate view-point configurations. If we cannot find a location that is unobstructed, we return one that is obstructed only by movable obstacles, as show in figure 14(d); HPN will then construct a plan to clear the view.

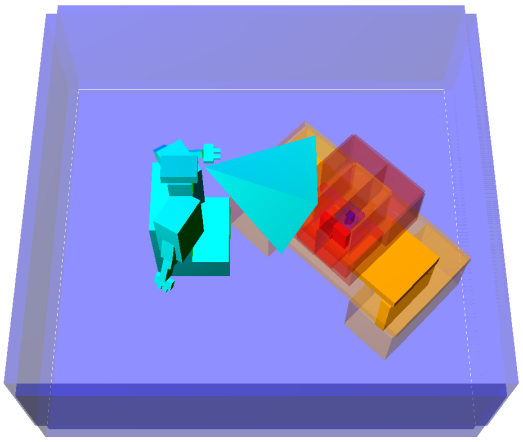
The generator then constructs a path from the home configuration to the view-point configuration, and also constructs a *view cone*, which is a frustum extending from the camera in the



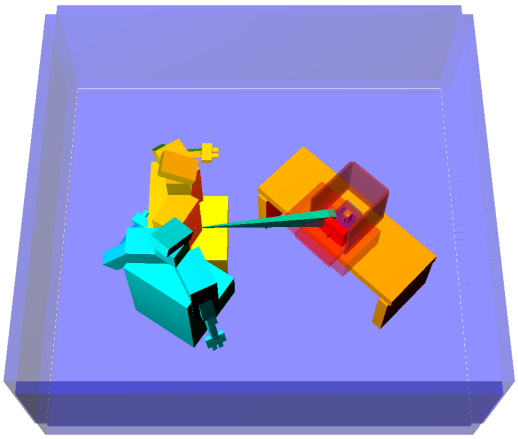
(a) Two-dimensional view of candidate look free space.



(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 intersects the red obstacle, which would need to be moved.

Figure 14: Candidate view locations used in generating look motions.

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 constraints and preferences are used:

- 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.

There are several 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 4.3.1, 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.

5.3.5 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. 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. This condition 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 can be use 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]**

If we do not know the universe of objects, but we are representing $\mathcal{S}_{obs}$ explicitly, we depend on a recursive decomposition of the region, illustrated in figure 15. 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 sub-regions, driven partly by the desire to aggregate space that has already been viewed into the same sub-region and space that has not been viewed into different sub-regions. For each of the sub-regions, 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}, γ)
    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]**

5.3.6 Clearing a region

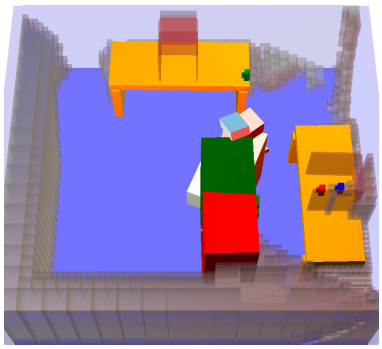
Coming to believe a region is clear is handled in a similar way. It currently makes use of a special region in space, called the *warehouse*, which is known to be "out of the way," in the sense that there are no goal object placements in that region. Having such a region is not crucial, and we could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

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: $\forall o \in occluders.\ BIn(o, warehouse \setminus r, \epsilon, \delta_{bc})$ **[2]**
    $BClearX(r, x \cup occluders, \epsilon)$ **[2]**

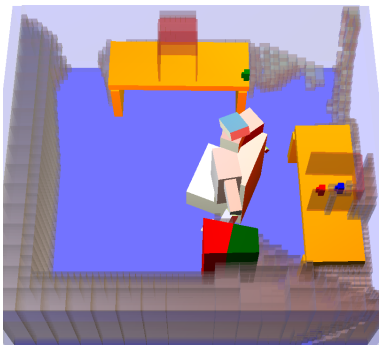
This operation is definitional and has no corresponding primitive. With abstraction value 1, it requires that the contents of r be known fairly accurately. At the next level, it generates a conjunction of requirements that all movable objects known to overlap the region that are not in the list of exceptions be in the part of the warehouse that does not overlap r. (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.

5.3.7 Putting an object in a region

We define an operator with no primitive that reduces a $BIn(o, r, \delta)$ condition to the conjunction of $PoseModeNear(o, \pi, \delta_{rp})$ and $BVRelPose(o, r, \epsilon_{rp}, \delta_{rp})$. There are generally many combinations of

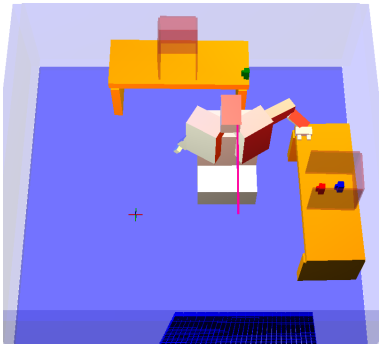


(a) Part of robot swept volume is split into two sub-regions (red and green).

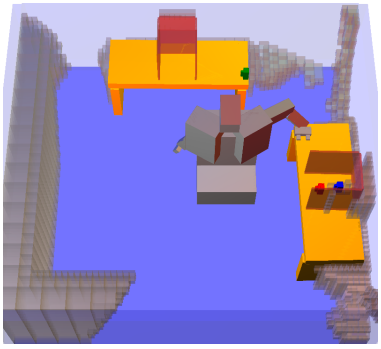


(b) The sub-regions are too big; the sub-regions are split again.

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



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



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

Figure 15: Determining the contents of a swept volume.

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 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 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 guarantee that the object is in this shadow with high probability, it is 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 PNM condition above. Then, we define EPSDELTASHADOW$(o, \epsilon, \epsilon_{rp}, \delta_{rp})$ to be SHADOW$(o, \mathbf{0}, \Sigma, \epsilon)$.

Now we have a shadow region for the object so that the 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 GENERATEPLACEPATH to find a placement for that shadow in r.

PUTIN$(o, r, \epsilon, b_{now}, \gamma)$:

effect: $BIn(o, r, \epsilon)$
let: $sh = \text{EPSDELTASHADOW}(o, \epsilon, \epsilon_{rp}, \delta_{rp})$
choose: $motion \in \text{GENERATEPLACEPATH}(sh, r, b_{now}, \gamma)$
pre: $PoseModeNear(o, targetPose(motion), \delta_{rp})$ **[0]**
$BVRelPose(o, r, \epsilon_{rp}, \delta_{rp})$ **[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 Example executions

In this section, we describe the results of executing the belief-space HPN method for pick-and-place problems on the PR2 robot. Extensions 1 and 2 (see Appendix A) are movies illustrating the real or simulated robot executions for the following examples. The simulated examples all include substantial initial state uncertainty and noise in motions and perception.

6.1 Detailed example of pick and place

Figures 16–19 show a slightly simplified HPN planning and execution tree for a simple example involving a blue baking-soda box (referred to as *soda*) and a red can (referred to as *soup*). Parts of the trees are labeled with upper case letters that are referred to in the text description. In the plan, the robot is referred to as *MandM*[8]. Blue nodes contain goals and green nodes contain primitive operations. For compactness, we use a single pink node to denote the steps of the plan that was made to satisfy the goal that is its parent. In many cases, pink nodes do not have as many children as they have plan steps: that is due to serendipitous achievement of some subgoals. The figures also include a sequence of snapshots of the belief state during planning and execution.

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 very high pose uncertainty.

The initial plan for placing the soda in the target region is shown in figure 16. At the highest level of abstraction, it is to (A) look at the soda, in order to know more precisely where it is and to plan a good method for picking it up, and then to put it in the target region. The planner finds a view cone that is likely to see the soda (a region of space defined relative to the soda that must be clear of obstacles) and constructs a plan (B) to be sure the view cone is clear, move to the viewing pose, and then look at the object. In the current belief state, it is possible that the soup and/or the table might intrude into the view cone, so the plan for coming to believe that the view cone is

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

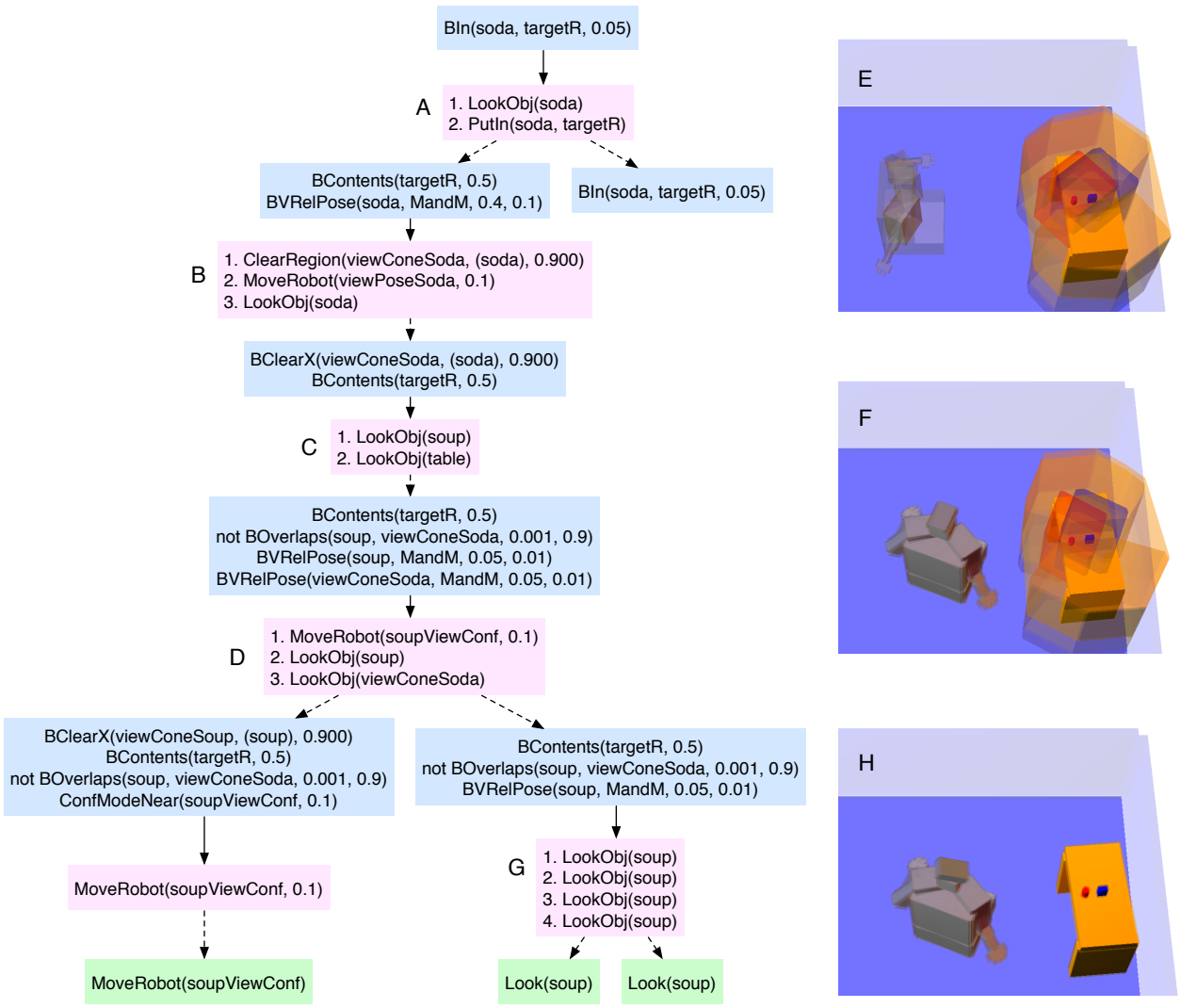


Figure 16: PR2 pick and place experiment: Uncertainty and initial look.

clear is (C) to look at both the soup and the table, with the goal of coming to know the relative pose of each of them with respect to the view cone for the soda.

To look at the soup, the robot plans (D) to move to a good viewing location for the soup and to look at both the soup and the view cone for the soda (recall this is a region anchored to the soda). Image (E) shows the robot's initial pose and uncertainty about the poses of the objects; image (F) shows the result of the first move, which brings the robot closer to the table and objects. At this point, it plans (G) to look at the soup four times, to dispel its pose uncertainty almost entirely. After two primitive looks, the pose of the soup is known sufficiently well, as shown in image (H), and BHPN execution returns to the level above. Here also we find that the view cone for the soda is well localized and, at the level above, that the table is well also well localized (because it was observed when looking at the soup). At this point, we are ready to achieve the second step in the high-level plan.

Figure 17 shows the next part of the planning and execution process. In order for the soda

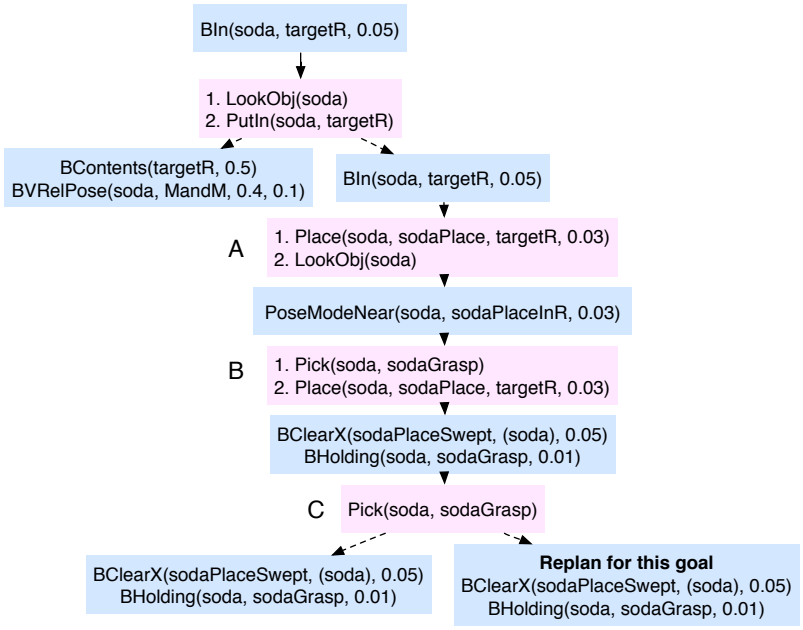


Figure 17: PR2 pick and place experiment: Beginning to plan to pick up the soda box.

to be believed to be in the target region, we must execute the plan (A) of placing the soda in a suitable place within the target region and then looking to verify that it is actually close enough to the intended pose. Placing the soda expands into a two-step plan (B) of picking it up with a grasp that is selected at this point and placing it. Next, we plan (C) to pick up the soda. We will explore that part of the tree in detail in the next figure, and also hint that it will not be entirely successful, and we will eventually have to revisit the goal to achieve plan C and make a new plan to achieve it.

Figure 18 shows the first attempt to pick up the soda. It is necessary (A) first to clear out the volume of space that the arm will sweep through in order to pick up the soda (based on a path that is generated here), and then to pick the soda itself. To clear that region we must (B) place the soup in a place that has been selected in the warehouse and look to be sure it has succeeded. Placing the soup requires (C), first that we pick it up with a selected grasp. Picking up the soup requires (D) that the robot move to a pose from which it can pick the soup only by moving its arm; we see the robot in this pose in image (E). Because of the base motion, the robot has accrued significant uncertainty about its pose with respect to the soup, so it plans to (F) look at the soup before picking it up. Looking requires (G) that it move the hand out of the field of view, as shown in image (H). Then, it observes the scene and the variances are decreased as shown in image (I). However, as a result of that observation, the belief distribution on the pose of the soda box is updated and it is found not to be near enough to the pose we had assumed it was in when the current plan was constructed. This constitutes a violation of the fluent $PoseModeNear(soda, sodaStart)$ and the BHPN control structure returns control to higher and higher levels until it reaches a point in which the current belief state remains in the envelope of the plan.

Figure 19 shows the rest of the planning and execution trace. Control was returned to point (A) in the tree, and a new plan is constructed for picking the soda, which generates a slightly different

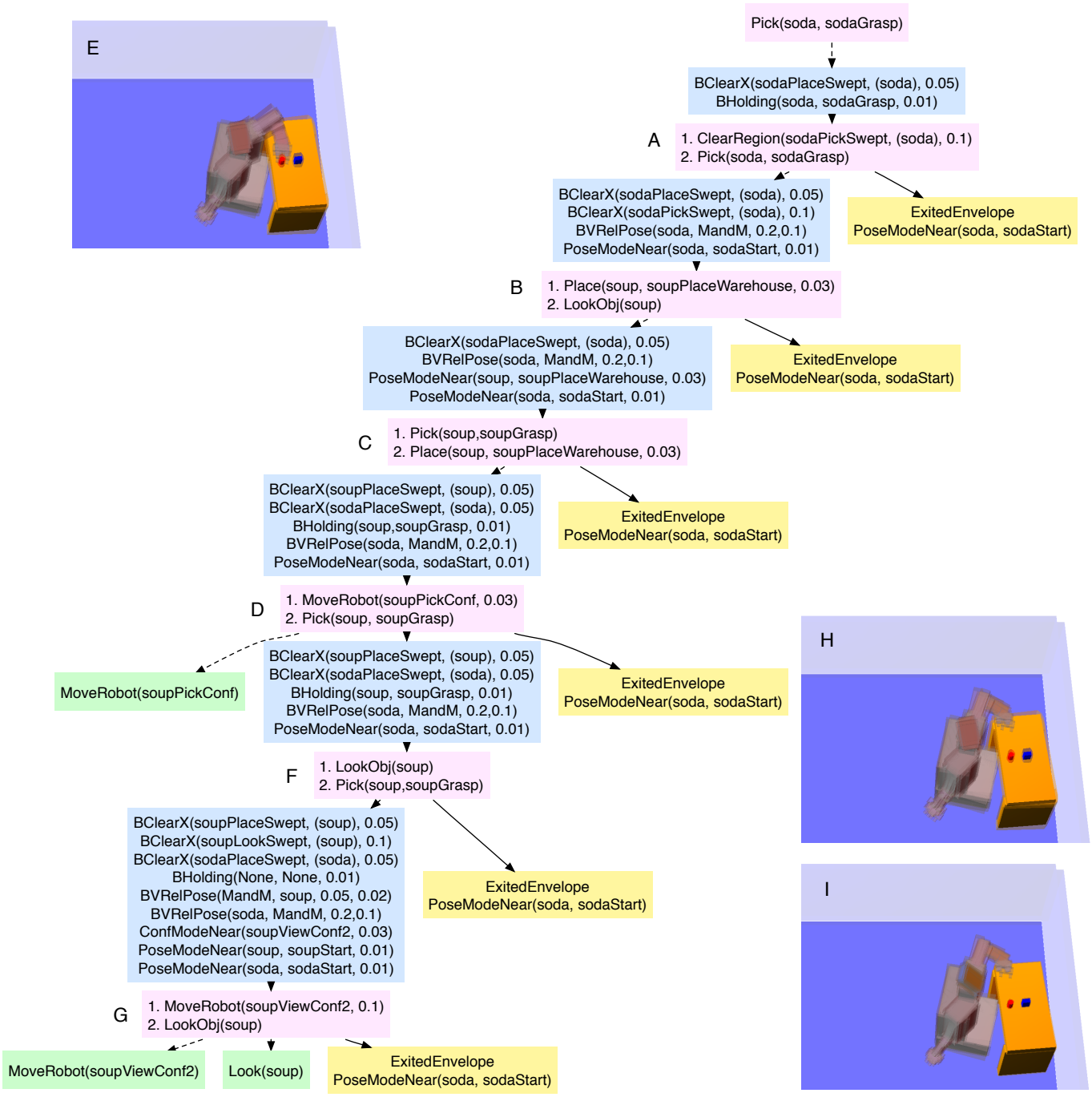


Figure 18: PR2 pick and place experiment: Realizing that the soup is in the way of picking up the soda box, and in the course of preparing to pick up the soup, finding that the soda has moved.

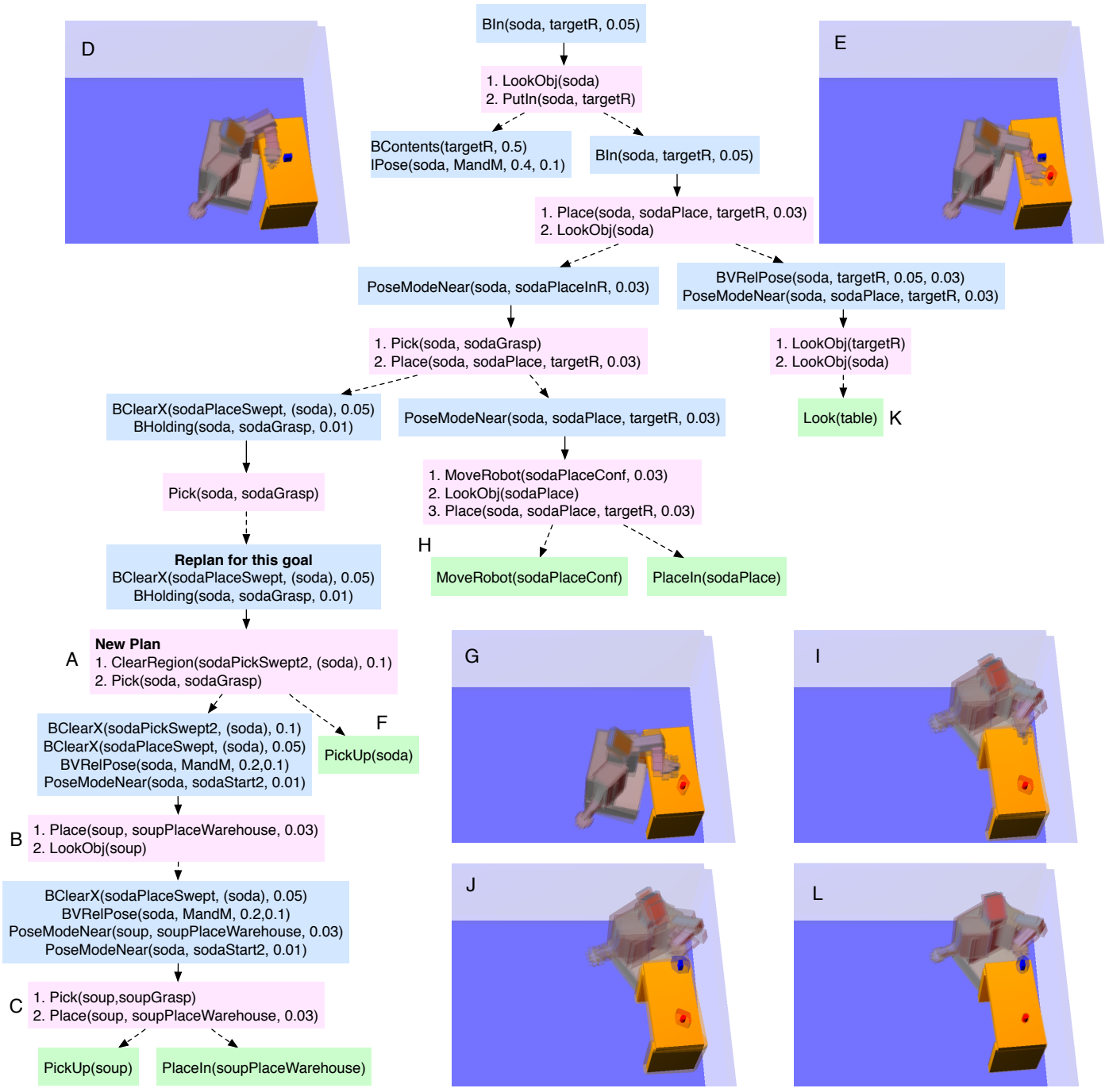


Figure 19: PR2 pick and place experiment: Action sequence that achieves the goal.

swept volume to clear. It is still necessary to (B) place the soup in the warehouse, for which it is necessary to (C) pick up the soup. In the current belief state, the objects are well localized with respect to the robot and so it executes a pick action, shown in image (D) and a place action, shown in image (E). The pose of the soup is moderately uncertain, but since the goal was just to get it out of the swept volume for picking the soda, there is no need to look again to verify its position. Because the robot did not need to move the base to execute these actions, it is still well localized with respect to the soda and so it (F) executes the pick action as shown in image (G). Now it moves the robot (H) and places the soda as shown in images (I) and (J). The mode of the soda's pose is satisfactory but there is too much pose uncertainty (due to the base motion) for the goal to be satisfied, so it performs a final look action (K), with the result shown in image (L).

Key frames in the execution of a similar planning and execution session are shown in figure 20. The total elapsed time, including planning, perception, and execution on the PR2 averages about 7 minutes. The planner, implemented in Python, takes about half of that time.

6.2 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. The average total execution time of each sequence (computed over 5 sequences) is approximately the same as that of the table-top experiment.

Figure 21(a) shows key frames from 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 21(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 several cycles of moving one object out of the way so as 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 improve the grasp strategy and the model of the uncertainty in the grasps. This would enable us to plan for actions that reduce the uncertainty either actively, by looking at the object in the hand, or passively, by exploiting slipping during placement.

- 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* [Karaman and Frazzoli, 2011] implementation which should reduce these problems. However, we should also never

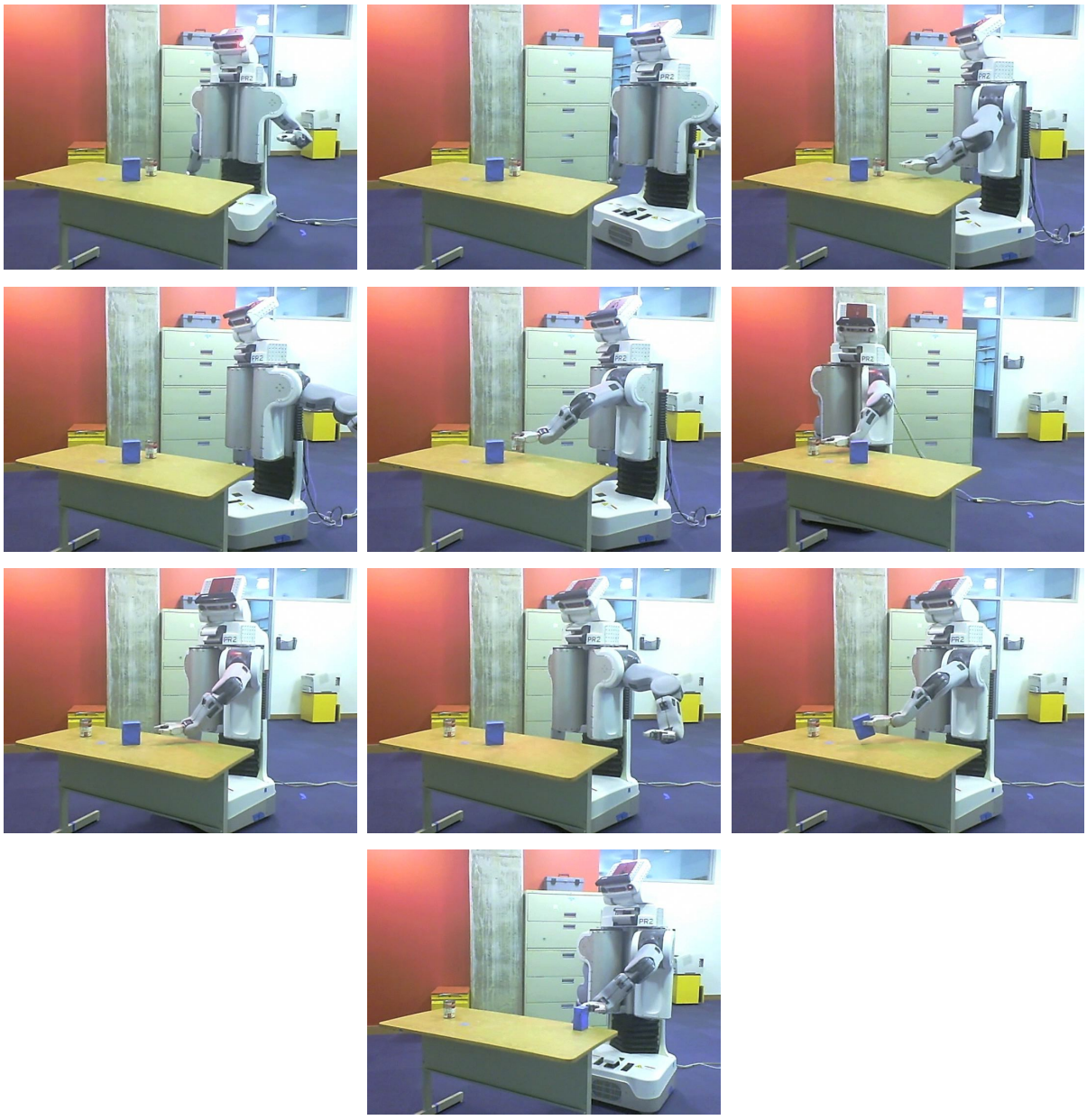
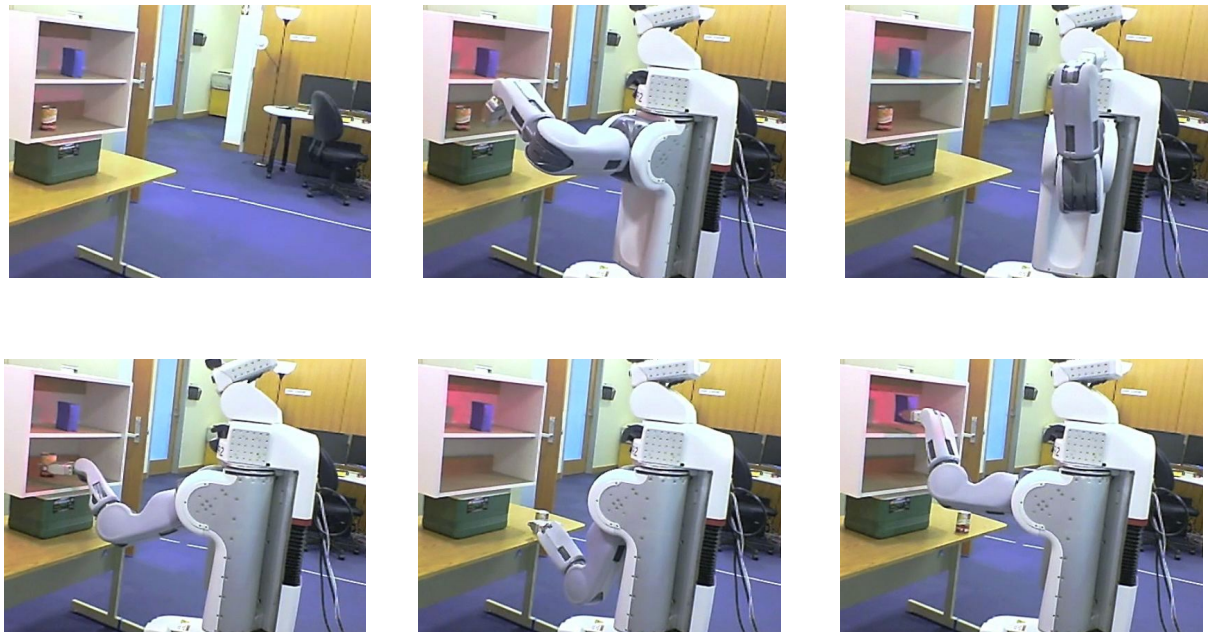


Figure 20: PR2 pick and place experiment: Execution



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

(b) Moving the soda box from the table to a region on the bottom shelf blocked by the soup 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 21: PR2 cupboard experiment

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.

6.3 Simulated examples using observed space oct-tree

A movie with simulations illustrating the version of the planner that uses the observed space oct-tree is included as Extension 3 (see Appendix A). It illustrates the robot explicitly choosing visual actions to clear out the space around it before it moves.

7 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 work raises some important questions. First, there is a question of how difficult it is to describe a new domain. It is, admittedly, a non-trivial problem to provide operator descriptions that result in robust behavior. We believe that the basic formalization of the pick-and-place domain is relatively general-purpose: it ought to apply almost directly to other robots. Given that basic capability, it is possible to build other domains on top of it. We have built upon this domain to make the robot search for objects in unknown cupboards [Wong et al., 2013], and to solve simple surveillance problems. In the completely observable case [Kaelbling and Lozano-Pérez, 2012a], we have built upon a basic pick-and-place capability to encode a kitchen domain involving cooking and washing dishes (in a somewhat abstract formulation). The additional operators to extend to the kitchen domain were relatively easy to construct. So, we believe that extensions to other applications might also be relatively straightforward.

Another important question is the domain of applicability of this method. If the domain is truly "puzzle-like," in the sense that there are many objects and little room to maneuver, then the hierarchical approach will not gain any traction. But, we believe that most real-world domains have enough room to allow hierarchical decomposition to have real utility. The belief-state representation here, a joint Gaussian, is admittedly naive: it scales poorly with the number of objects in the world. It is an important area of future work to explore more highly decomposed and non-analytic (particle-based) representations of belief, and to characterize operator pre-images in those representations. This is a critical area of future research, but we are optimistic that it will be possible to represent a wide variety of types of uncertainty over very large domains.

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

This work was supported in part by the NSF under Grant No. 1117325. Any opinions, findings, and con- clusions 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 FA2386-10-1-4135 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.

We thank all the members of the LIS research group for their help in this project, and in particular: Jared Glover, Aaron Fryman, Will Grathwohl, Sanja Popovic and Dawit Zewdie for developing the 3D perception subsystem; Lawson Wong for the UKF implementation and constrained pose estimation; Ashwin Deshpande and Dylan Hadfield-Menell for help with hierarchical planning; Alejandro Perez, Matthew Jordan, and Rodrigo Gomes for help with motion planning; Jennifer Barry for help in testing the planner and with PR2 software; and Caelan Reed Garrett for help with Cython. We also thank Martin Levihn for carefully reading the manuscript.

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 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.

Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4), 2003.

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.

Blai Bonet and Hector Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

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*, 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*, 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, 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.

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 robot task and motion planning in the now. Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2012a.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. Technical Report 2012-019, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2012b.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

Emil Keyder and Hector Geffner. The HMDP planner for planning with probabilities. In *Sixth International Planning Competition at ICAPS08*, 2008.

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. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32:108–120, 1983.

Tomás Lozano-Pérez and M. A. Wesley. An algorithm for planning collison-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570, 1979.

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.

Mausam and Andrey Kolobov. *Planning with Markov Decision Processes*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

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.

Joseph M. Romano, Kaijen Hsiao, Günter Niemeyer, Sachin Chitta, and Katherine J. Kuchenbecher. Human-inspired robotic grasp control with tactile sensing. *IEEE Transactions on Robotics*, 27(6), 2011.

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.

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 Goncalves, and Adelardo 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.

Lawson L.S. Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *IEEE Conference on Robotics and Automation*, 2013.

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 Index to Multimedia Extensions

The multimedia extensions to this article are at: `http://www.ijrr.org`

Extension	Type	Description
1	Video	The PR2 performing tabletop pick and place, as described in section 6.1.
2	Video	The PR2 performing cupboard pick and place, as described in section 6.2.
3	Video	Simulated PR2 using observed space oct-tree, as described in section 6.3.