2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Implicit Belief-Space Pre-images for Hierarchical Planning and Execution

Leslie Pack Kaelbling and Tomás Lozano-Pérez

***Abstract*— We present a method for planning and execution in very high-dimensional mixed discrete and continuous spaces in the presence of uncertainty using an *implicit, factored approximation* representation of pre-images and extend it to planning in belief space. We demonstrate the approach in a mobile-manipulation domain combining pushing with pick-and-place manipulation with error in sensing and manipulation. We show empirically that execution monitoring using pre-images improves computational efficiency over continual replanning, and that the hierarchical planning method it enables provides further efficiency improvements.**

I. Introduction

One approach to solving very large, long-horizon, uncertain planning and execution problems is embodied in the BHPN [1] system, which uses hierarchical planning, replanning, and execution to obtain robust behavior with manageable time requirements. Critical to this strategy is the ability to describe and manipulate representations of plan pre-images, which will serve as subgoals for hierarchical planning and as validity conditions for execution monitoring.

The *pre-image* of a goal with respect to an action or plan is a fundamental concept that underlies regression-based planning algorithms, enables execution monitoring, and supports goal-based hierarchical planning. A goal Γ is a set of system states; the preimage of Γ under an action sequence is the set of system states Γ' such that executing that sequence of actions from any state in Γ' will cause the system to transition to a state in Γ.

The planning algorithm known as *pre-image backchaining* in the robotics community [2] and *goal regression* in the AI planning community [3] operates by searching in the space of pre-images: the starting node in the search tree is the goal, intermediate nodes are the pre-image of their parent node under some action, and the search terminates when a pre-image is reached that contains the initial state of the system.

Execution monitoring allows a system to execute plan steps until a state is reached in which the plan is no longer valid, and then replan. The pre-image of the goal under any suffix of the plan is the set of states in which that plan suffix will achieve the goal. Thus, a straightforward and effective plan execution and monitoring strategy is to find the shortest plan suffix whose pre-image contains the current system state and execute its first action. If no pre-image of the plan contains the current state, then a new plan must be constructed and executed with this same monitoring strategy.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA lpk@mit.edu, tlp@mit.edu. We gratefully acknowledge support from NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

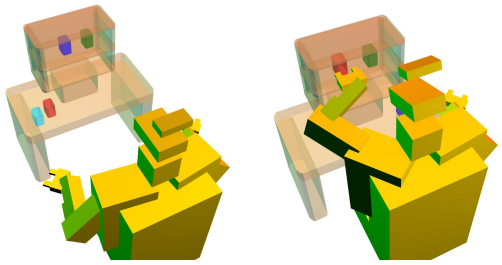


Fig. 1: Initial and final state for shelves example.

Ideas of online execution monitoring can be extended to a *hierarchical planning system* [4], [1] that constructs plans in a hierarchy of abstracted domain models, planning in more detail only for the first "step" of an abstract plan. Pre-image-based planning plays two critical roles in this process. First, the results of abstracted operators cannot be modeled in the system-state space so we plan via regression using subsets of the actual preconditions. Second, the pre-images of the steps in an abstract plan serve as goals for the hierarchical planner at the next, more concrete, level of abstraction.

In discrete domains, represented using STRIPS or PDDL operator descriptions, state-sets (including goals and pre-images) are represented using sets of logical *fluents* that specify values of some aspects of the state. Given those symbolic operator descriptions, the explicit pre-image of a state-set under an action can be computed very straightforwardly. In continuous domains, such as robot manipulation planning, it is difficult both to represent and to compute explicit pre-images [2], [5].

We construct a method for planning and execution in very high-dimensional mixed discrete and continuous spaces in the presence of uncertainty, based on an *implicit, factored approximation* of pre-images in continuous spaces and extend it to apply to the case where the space is actually the "belief space" of probability distributions over underlying world states. We provide a planning algorithm that searches in the space of pre-images in this representation. Finally, we demonstrate this approach in a mobile-manipulation domain that combines pushing with pick-and-place manipulation using actions with motion and sensing error.

The approach presented here is related to our earlier work on the Hierarchical Planning in the Now (HPN) system [4], [1], [6]. The two key differences are: (1) the representation of pre-images by using implicit fluents such

as $CanReachHome$ and $CanPlace$ instead of using explicit representations of swept volumes of particular paths as obstacles, and (2) the introduction of a general notion of *conditioning* in the regression algorithm to handle these implicit fluents. Together these extensions generalize and make systematic the pre-image computation approach in our earlier work. Our contribution in this paper is to show how to tractably plan using implicit representations of pre-images in hybrid state-spaces and to demonstrate their use for hierarchical planning and execution monitoring in real robot manipulation problems.

We show empirically that execution monitoring using pre-images provides a substantial improvement in computational efficiency over continual replanning, and that the hierarchical algorithm enabled by abstract pre-image backchaining provides even further efficiency improvements.

II. Related work

Planning under uncertainty can be done optimally by making conditional plans offline [7], [8]. For efficiency and robustness, our approach is to construct a deterministic approximation of the dynamics, use the approximate dynamics to build a plan, execute the plan while 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 [9], [10], [11] as well as symbolic planning domains [12].

Execution monitoring in robotics has a long history; see [13] for a survey. In particular, the use of goal-regression (pre-images) for execution monitoring dates back to Shakey's PLANEX [14], although the formalization of this strategy appears to be much more recent [15]. However, previous work in this area has been limited to purely symbolic STRIPS operators where computing pre-images is relatively simple. Computing pre-images for geometric problems dates back to the "pre-image backchaining" paper [2]; excellent summaries of subsequent work on planning with uncertainty are available [16], [17]. However, computing such pre-images exactly is known to be intractable [5].

Manipulation planning is a hybrid (multi-modal) planning problem involving both continuous actions and discrete choices. This type of planning problem can be attacked using extensions to classic randomized motion planning algorithms [18], [19], [20], but these approaches tend to suffer from a lack of goal guidance. Many recent approaches to manipulation planning integrate discrete task planning and continuous motion planning algorithms. Many of these approaches use a motion planner as a way of verifying plans produced by a task planner [21], [22], [23]. Others combine the task planner and motion planner more intimately allowing for the motion planning to influence the task planning and vice versa [24], [25], [26], [27], [28].

III. Factored, conditional pre-images

In this section we use a sequence of *highly simplified* example domains to illustrate the concepts underlying the implicit representation of pre-images and their role in analyzing the robustness of plans.

One-dimensional example In the first example, objects with fixed extent are placed in a one-dimensional space. Objects cannot overlap with one another. There is a point robot that may move freely along the line. The robot may be attached to an object, in which case, it moves when the robot moves. When the robot is holding an object, it is held "above" the other objects and so can move past them without collision. The robot can "pick" any object if the robot's configuration (place along the line) is within the extent of that object. It can "place" an object if it is holding that object and the part of the line that the object will be placed onto is "free" (does not contain any other objects). In the case of a single object, there are two continuous state variables, the "configuration" c of the robot and the "pose" p of the object (the position of its center), and a discrete state variable h indicating whether the robot is holding the object. Both c and p take real values and h is Boolean.

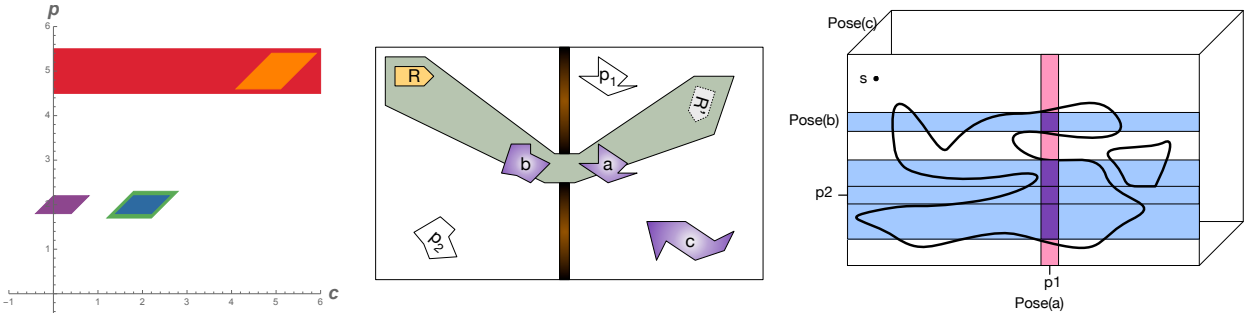
We begin by considering a problem in which the goal Γ is for the object to be at position 5.0; the object starts at position 2.0 and the robot at position 0.0. The object has extent 1.0. Because we are in a domain with continuous actions, there will be, in general, an infinite number of feasible plans; this makes the planning process complex but does not impact the analysis of the pre-images of a particular plan. For concreteness, we will consider the plan (where the argument to $move$ is a displacement)

$$Move(1.8); Pick; Move(3.0); Place$$

The pre-image for p in this plan has no "volume": if the object is not precisely at 2.0 then the plan will not work. However, because the object may be grasped anywhere along its length, there is tolerance to variation in the initial configuration of the robot. It is never sensible for the goal for a real system to be a point value in real space. Consider the previous example, but where Γ is $p \in [4.5, 5.5]$, where p is the position of the left edge of the object. Now there will be larger tolerances on the initial state of the overall plan. The "grasping" constraint takes the form of a relationship between the robot and the object, so it is easier to specify pre-images in terms of $p - c$ and p below.

In general, there will be error in execution of actions. More robust plans can be obtained by planning with an explicit model of the error induced by the actions. Let us assume that whenever the robot attempts to move by u, in fact it moves by an amount in $[u - 0.1, u + 0.1]$ and, similarly, that whenever it picks or places an object, the pose might be perturbed by as much as 0.1 in either direction relative to the robot, but the robot does not move. Here are the pre-images of the plan steps, also shown in figure 2a.

Action	$p - c$	p	h	color
Goal		$[4.5, 5.5]$		red
$Place$	$[-0.5, 0.5]$	$[4.6, 5.4]$	**True**	orange
$Move(3.0)$	$[-0.5, 0.5]$	$[1.7, 2.3]$	**True**	green
$Pick$	$[-0.4, 0.4]$	$[1.8, 2.2]$	**False**	blue
$Move(1.8)$	$[1.5, 2.1]$	$[1.8, 2.2]$	**False**	purple



(a) Preimages for 1D domain. (b) Two-dimensional domain. (c) Object configuration space

Fig. 2: Illustrative examples.

The final pre-image has volume, but it is considerably smaller than it would have been in a domain with perfect actions.We can also see that if there had been more steps in the plan with similar error, the pre-image would have collapsed, rendering the plan invalid. Taking errors explicitly into account when planning also means that a plan with a different structure but less cumulative error might be selected by the planner.

In most cases, it is impossible even to bound the error of a physical system, with complete certainty. More generally, we will have a distributional model of the system's errors. In many such cases, it is possible to provide a bound on the error of any given action, with some probability p. Generally, there will be a trade-off: a smaller error bound can only be guaranteed with a smaller probability. A reasonable criterion for selecting the sizes of error intervals is to select a plan with maximum success probability (the product of the probabilities associated with each of the actions in the plan) such that the pre-image contains the actual initial world state. This criterion will put larger intervals places that makes the entire plan as likely as possible to succeed.

Factored representation of intervals In more complex domains, with many objects and subtle interactions, the geometric descriptions of the pre-images become increasingly complex as they are propagated backward through action sequences. Simple linear constraints are insufficient to represent them exactly and algorithms for computing them are difficult. Our approach will be to trade approximation for representational and computational tractability.

One step of representational simplification is to *factor* the descriptions of pre-images into conjunctions of interval constraints on individual variables. For example, consider the set from our simple example: $p \in [4.5, 5.5], p - c \in [-.5, .5]$. In p, c space, it does not form a rectangle, so it cannot be exactly represented in the form $p \in i_1, c \in i_2$. In order for a regression-based planner to be correct, it is enough to compute approximate pre-images that are *subsets* of the true pre-images. These conditions are sufficient to guarantee that the plan will execute correctly but not necessary: the might be correct in states that are not in the approximate pre-image.

There are many rectangles that can be inscribed in the shape characterized by $p \in [4.5, 5.5], p - c \in [-.5, .5]$; we would like the largest one, but even that is not uniquely specified. We will generally use the largest axis-aligned hypercube whose center is the same as the center of the exact pre-image. In our example, that is defined by $p \in [4.75, 5.25], c \in [4.75, 5.25]$. In domains where the errors in actuation are asymmetric, for example, we might wish to select a hyper-rectangle that has more tolerance in some dimensions than others. In realistic high-dimensional domains, many variables will be irrelevant and, hence, completely unconstrained.

Implicit representation and conditioning The previous example domain involved only conditions on individual dimensions of the configuration space or simple relations between them. In interesting manipulation-planning problems, the combined configuration space of the robot and all the objects is very high-dimensional and the important constraints may involve all the variables. Consider the domain shown in figure 2b. It shows a two-dimensional workspace containing a robot with three degrees of freedom (x, y, rotation) at its initial (home) position R; three movable obstacles (a, b, and c) each also with three dof; and a desired final configuration of the robot, R'. The dark rectangles are immovable obstacles. The green region is the swept volume of an example of a possible path for the robot from R to R'.

The overall configuration space of this domain is 12-dimensional so we can only visualize it abstractly. Consider the pre-image of a collision-free motion of the robot from its initial (home) configuration R to R'. That pre-image is the set of possible joint placements of a, b, and c such that there exists a path between R and R'. It is nearly impossible to characterize its shape exactly and explicitly. Our approach to pre-image backchaining relies on an implicit representation of these sets and limited computations on them.

Figure 2c shows a "cartoon" drawing of the combined configuration space of objects a, b, and c; each dimension in this diagram represents the three-dof pose of one of the objects, and the dimensions corresponding to the robot are not represented. The interior of the dark black outline represents the set of object configurations that collectively have the property that the robot can reach its home configuration R from R' via a collision-free path; we will denote this set as $\text{CRH}(R')$ (CRH stands for $CanReachHome$). The current world state, denoted s, is not inside the black contour, which means no collision-free path is available. This is something that we can detect (with high probability) via the failure of a motion-planning query.

It is not immediately clear how to do regression-based

planning in this situation, because no individual operation can drive the objects into a configuration that satisfies $\text{CRH}(R')$. Assume that the robot is able to move individual objects to new poses. We might consider moving object a to a new pose, p_1. This suggestion can be arrived at by finding a pose of a that does not overlap the swept volume of some robot path that allows obstacle collisions, but tries to minimize the number of collisions [29], [30]. We could then rewrite $\text{CRH}(R')$ to a conjunction

$$Pose(a) \in (p_1 \pm \delta) \;\&\; \text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta)\})$$

The first term asserts that the pose of a is near p_1, which is shown as the vertical pink stripe in the figure. The formula $\text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta)\})$ is a *conditional* expression: it asserts that *if a were near pose p_1* and everything else was the same as it is in a state s, then $\text{CRH}(R')$ would hold in s. This set consists of the two horizontal blue bands in the figure: in any combined object configuration in those regions, if we were to set the pose of a to be near p_1, then the state would be in the set satisfying $\text{CRH}(R')$. So, the intersection of the blue set and the pink set (shown in purple) represents the conjunction of the two conditions, which is a subset of the pre-image of the goal (robot at R') under the plan $Move(R')$; the blue set is the pre-image of the goal under the plan $Place(a, p_1); Move(R')$.

The initial state is not yet in the blue set (which we can check via another motion planning query in a hypothetical world where a is at pose p_1). However, we can expand the pre-image of the whole plan by adding an operation to move b near to pose p_2; this pose can be suggested by avoiding the swept volume of a path that reaches R' in a world where we ignore collisions with b but have a at pose p_1. Then, we can rewrite the formula $\text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta)\})$ to

$$Pose(b) \in (\mathrm{p}_2 \pm \delta) \;\&$$
$$\text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta) \& Pose(b) \in (p_2 \pm \delta)\})$$

The first term represents the small horizontal stripe within the blue region, aligned with p_2 on the vertical axis; the second term represents any point that is vertically above or below the blue region, which in this case is the whole plane (for this particular value of $Pose(c)$); this second set contains the initial state.

The process we have followed is one of regression. Assume the current state is $s = (s_a, s_b, s_c)$ where each component is a pose for an obstacle. Table I illustrates the regression process. The first column shows the ultimate goal, and then the sequence of pre-images of that goal under actions in the plan; subsequent columns point out the relevant aspects of the figure, show the geometric query for membership in the pre-image, indicate how the parameters of an action are generated, and show the action that will be taken to achieve the goal on that line. In general, there must be a search in the space of orderings of these operators, as only some orderings will result in feasible plans.

This demonstrates pre-image backchaining in a hybrid space without an explicit characterization of the pre-images. It is only necessary to test whether the initial state is in the pre-image (a motion planning query in a hypothetical world) and to suggest actions that expand the pre-image (for example, moving an objects out of the way). The conditioning process tracks the assumptions that are needed to construct the relevant hypothetical worlds.

IV. Representation

In order to build a planner that searches in the space of pre-images, we need a representation for pre-images and for the dynamics of the domain. We will use ideas from classical AI planning, augmented to apply to hybrid domains with mixed discrete and continuous state and action spaces. These representations are general-purpose and can be applied to state space or belief space.

Fluents: In order to plan in domains with large (or possibly unbounded) numbers of objects, rather than enumerating the set of state variables that describes the domain, e.g. for the pose of each object, we use logical expressions, called *fluents* because their distributional values change over time, to name them. A fluent is made up of a symbolic *predicate* and one or more symbolic *arguments*. When constant values (denoting objects, regions of space, robots, etc.) are supplied as arguments to a fluent, it is called a *ground fluent* and it denotes a state variable. Fluents may have discrete or continuous values. Every ground fluent has a single value in a state; we will write $V(\phi, s)$ to stand for the value of fluent ϕ in world state s. So, the statement $\phi = v$ *holds* in world state s if and only if $V(\phi, s) = v$. We define three types of fluents.

Primitive fluents directly characterize state variables in the underlying domain; the union of all possible primitive fluents provides a complete specification of a world state. The semantics are straightforward. We can think of s as being a (possibly indeterminate length) vector of values and any primitive fluent ϕ as being an "index" into that vector, so that $V(\phi, s) = s_\phi$. *Implicit fluents* characterize aspects of the underlying state, about which we might wish to make assertions; their values are functions of values of the primitive fluents. The $CanReachHome$ fluent is an example. Associated with each implicit fluent ϕ is a function f_ϕ that maps the state into a value, so that $V(\phi, s) = f_\phi(s)$. *Conditional fluents* are implicit fluents with an additional argument that consists of a list of primitive fluents with assigned values, which must not be in contradiction (that is, specify conflicting values for the same fluent). Intuitively, the semantics is that the implicit fluent would hold in the state if it were modified just in the dimensions required by the primitive fluents. A conditional fluent has the form $Q(a_1, \ldots, a_n; \phi_1 = v_1, \ldots, \phi_k = v_k)$, where Q is the predicate, a_i are the arguments of the fluent, and $\phi_j = v_j$ are the fluent values being conditioned on.

Define $C(s; \phi_1 = v_1, \ldots, \phi_k = v_k)$ to be a hypothetical world state, in which $s_{\phi_i} = v_i$ for $i \in 1, \ldots, k$. Then,

$$V(Q(a_1, \ldots, a_n; \phi_1 = v_1, \ldots, \phi_k = v_k), s) =$$
$$V(Q(a_1, \ldots, a_n), C(s; \phi_1 = v_1, \ldots, \phi_k = v_k))$$

Goal	Visual	Geometric query	Gen	Action
$Conf = R'$	robot at R'	$R = R'$		$Move(R, R')$
$\text{CRH}(R')$	black contour	$path(R, R')$ in (s_a, s_b, s_c)	p_1 so a can be avoided	$Place(a, p_1)$
$\text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta)\})$	blue region	$path(R, R')$ in (p_1, s_b, s_c)	p_2 so b can be avoided	$Place(b, p_2)$
$\text{CRH}(R', \{Pose(a) \in (p_1 \pm \delta), Pose(b) \in (p_2 \pm \delta)\})$	slice with s_c	$path(R, R')$ in (p_1, p_2, s_c)		

TABLE I: Regression planning process

where we assume a function $f_{Q(a_1,\ldots,a_n)}$ that can supply the unconditional value of fluent $Q(a_1, \ldots, a_n)$ in a state.

Operator descriptions: We assume that system has a finite set of primitive action types, each of which may have zero or more discrete and/or continuous parameters. The dynamics of these actions is described using operator descriptions, which are parametrized so as to offer a compact representation of the effects of infinitely many action instances on domains of arbitrary size.

A critical assumption underlying this strategy for describing world dynamics is that each individual action instance only affects the values of small set of fluents and that the rest are unchanged or that, if they are changed, those changes can be relatively systematically specified. An operator description has the form:

NAME(θ, θ'):

result:	$\phi_1(\theta) = v_1, \ldots, \phi_k(\theta) = v_k$
gen:	$\theta'_1 \in g_1(\theta), \ldots, \theta'_m \in g_m(\theta, \theta'_1, \ldots, \theta'_{m-1}),$
precond:	$\psi_1(\theta, \theta') = u_1, \ldots, \psi_l(\theta, \theta') = u_l$

The θ and θ' arguments are vectors of variables; the ϕ_i and ψ_i are fluents that may have constants or elements of θ, θ' as arguments; the v_i, u_i are either be constants or elements of θ, θ'. The operator is implicitly universally quantified over values of the variables θ. When an operator is applied during the search, variables in θ are bound by matching the operator's results to fluents in the goal. There may be additional variables in θ' that are not yet determined; these represent the variety of ways in which the operation can be carried out to obtain the same result.

Although the operator is defined for any binding of values to the variables in θ', most choices of bindings will result in an operation that is unhelpful or or incompatible with other aspects of the system's goals. So, an operator description includes *generators*, g_i: given a vector of values of variables that were bound in the operator application or by previous generators, a generator g is capable of generating one or more possible values of a parameter in θ', such as a grasp, object placement, or path; formally, it behaves like a function that can be called multiple times to obtain multiple values, until it is exhausted.

Implicit fluents and conditioning: In most planning formalisms, the preconditions of each action in a plan must be established by matching them to result fluents of other actions in the plan, or to fluents that are true in the initial state. In our domains of interest, there are fluents such as $In(o, r)$ asserting that an object is in some region of space or $CanReach(c_1, c_2)$, asserting that the robot can move from configuration c_1 to configuration c_2 without collision.

It would be impossible to assert all possible fluents that change value as a result of taking an action. These *implicit* fluents appear as preconditions, but not as results of actions. To allow pre-image backchaining to succeed, we must make a connection between implicit fluents occurring in a goal and operations that can possibly achieve them. We do this by introducing *inferential operators*, which have implicit fluents as a result, and primitive fluents as preconditions. We might have a situation in which a $CanReach(c_1, c_2)$ fluent is in the goal, but is not true in the initial state, because an object x is in the way. An inferential operator is shown below: ACHCANREACH(C_1, C_2, O, P, X, X'):

result:	$CanReach(C_1, C_2, X) =$ **True**
gen:	$(O, P) \in AchCanReachGen(C_1, C_2, X)$
	$X' \in AddCondition(X, Pose(O) = P)$
precond:	$CanReach(C_1, C_3, X') = True$
	$Pose(O) = P$

The result of this operator is a conditional $CanReach$ fluent, which has two configurations as arguments, as well as a list of conditions X. A generator, $AchCanReachGen$, considers both the arguments and the initial state and suggests that if object O were at pose P, the $CanReach$ fluent would become true or would at least become easier to make true. The preconditions of the inferential operator are that the object be at the suggested pose and that the $CanReach$ fluent be true, conditionally, if O were in location P.

Conditional fluents are a representational mechanism for the techniques in table I that allow us to specify pre-images implicitly without characterizing them explicitly in terms of huge (or infinite) disjunctions of conjunctions of primitive fluents (the set of arrangements of objects that allow configuration c_1 to be reached from c_2 is very complex). In addition, they enable inferential operators to make the connection between implicit preconditions and operators that can help achieve them, by making selective aspects of those preconditions explicit in terms of primitive fluents.

V. PLANNING ALGORITHM

Planning takes place in the context of the BHPN system [1], which performs hierarchical planning and execution, based on a regression planner. Pre-images computed during the regression at one level are used as subgoals for lower levels of hierarchical planning, as well as to select the next plan step for execution, or even to decide to abandon the current plan entirely and re-plan at a higher level of abstraction.

The regression planning algorithm takes an initial domain state s_o, which may be have any representation, as long as each fluent can be tested to see if it holds in that state, a set of operator descriptions and a goal as input and returns a plan. It searches in the space of pre-images, which are conjunctions (represented as sets) of fluents. It uses the A^*

algorithm, with the domain goal as the initial state of the search process, and terminates when it reaches a G_0 such that for all fluents $(\phi_i = v_i) \in G_0$, $V(\phi_i, s_0) = v_i$.

To compute the successors of a state in the search we compute the pre-image of subgoal G under each applicable operator O. The procedure REGRESS (slightly simplified pseudo-code is shown below) does this; G is a conjunction of fluents in the goal, O is an operator that can have as its effect at least one fluent in G and has some of its parameters already bound to match that effect, and s_0 is the initial world state. If successful, it returns o, an instance of O with all parameters bound and pre-image G'. It begins by "discharging" any fluents in G that are made true by O. In addition, it adds the result fluents to the conditions of any conditional fluents; if these conditions already contain a value for a result fluent, then the existing value is retained. Next, it calls the generators $O.gen$ to get a set of bindings for the remaining unbound variables $O.\theta'$, and applies those (application is denoted by the $\backslash$ symbol) to operator O to get an operator instance o with all of its variables bound. The preconditions of o are added to G', which is then subjected to some simple tests for inconsistency and infeasibility. If it passes, then G' is the pre-image of G under o and the pair (o, G') is returned.

In the high-level search, each goal G is returned to the search agenda after expansion, with a record made of which ground operator instances have been applied to it; to retain completeness of the search in infinite action spaces, it may be re-extracted and re-expanded using new bindings for θ'.

REGRESS(G, O, s_0) :
1 $G' = \{\phi \in G \mid \text{not } O.result \text{ implies } \phi\}$
2 **for** $\phi \in G'$:
3 **for** $r \in O.result$: $\phi.addCondition(r)$
4 $o = O\backslash$GETBINDINGS$(O.\theta', O.gen, G', s_0)$
5 $G' = G' \cup o.preCond$
6 **if** G' is inconsistent: **return None**
7 **return** (o, G')

If a precondition is inconsistent with G', then the regression step fails, which slows down the search. Generators should yield values θ' that produce operator instances o that: do not make the conditional fluents in G' infeasible when their results are added to the conditions; have preconditions that are consistent with all fluents in G'; and, preferably, have preconditions that are easy to achieve from s_0.

The factored structure of the goal representation is also critical for efficient computation of a domain-independent heuristic, which efficiently computes an estimated cost by considering elements of a conjunction independently.

VI. MOBILE MANIPULATION DOMAIN

We applied this planning algorithm to a domain in which a PR2 robot manipulates boxes in an environment with tables and shelves. The robot is uncertain about the poses of the objects, and the robot's base motions introduce substantial new uncertainty. An object detector, operating on point clouds from a Kinect sensor, produces noisy estimates of the object poses. We employ a state estimator that, whenever the actual robot takes an action and receives an observation, performs a Bayesian update on a representation of a distribution representing the agent's current belief about the underlying state of the world. The planning is carried out by pre-image backchaining in belief space.

We use operator descriptions to describe this POMDP, but not as it is typically done. Rather than specifying the stochastic transition dynamics of the world-state process and the observation model, we directly describe the dynamics of the belief-state process itself. We model the entire "plant", including the state estimator, that takes actions as inputs and generates belief states as outputs. The goal of the overall system is to select actions to drive the system into a belief state in the goal set. Actions without observation, such as moves, are deterministic in belief space even if stochastic in state space. Observations are stochastic in belief space; we plan in a deterministic approximation that assumes the most likely observation will be obtained.

Pre-images in belief space Pre-images in belief space are sets of belief states; we characterize these sets using a conjunction of constraints, each articulating a condition on the distribution of a random variable.

The primitive fluents for the mobile manipulation domain correspond to the configuration of the robot, the poses of objects, and the pose of grasped objects in the gripper. We define *belief fluents* that describe constraints on the distributions of an underlying primitive fluent, ϕ. Belief fluents for continuous-valued fluents specify a probability p, a variance σ and a tolerance δ. Such fluents are satisfied by any Gaussian mixture distribution of ϕ in which there is a component with mixture weight $\geq p$, mean $\in [v - \delta, v + \delta]$, and variance $\leq \sigma^2$. This idea can be extended to multivariate Gaussians and to Gaussians in tangent space to handle quantities such as rotations. These fluents are written $B(\phi, v, \sigma, \delta, p)$ For discrete variables, the belief is specified by a value v and a probability p. Such fluents are satisfied by distributions that assign a probability $\geq p$ to the event that the condition has value v. These fluents are written $Bd(\phi, v, p)$; for binary-valued fluents, we drop the v, and assume $v = True$.

We use beliefs over implicit conditional fluents to represent the existence of a collision-free path from a given robot configuration to a "home" configuration ($CanReachHome$); the existence of a collision-free path between configurations that share a base location ($CanReachNB$); the feasibility of picking or placing an object at a given pose from a given robot configuration ($CanPick$, $CanPlace$); and the feasibility of pushing an object to a given pose from a given robot configuration ($CanPush$).

Generators Implementing the planning algorithm for the mobile manipulation domain requires constructing generators that instantiate the free variables in the operator specifications. In this domain, the generators (a) plan robot paths, (b) draw from a precomputed set of grasps, (c) sample poses for objects and (d) sample robot configurations where an object is in view of the Kinect sensor. In all cases, these

choices must avoid collisions with immovable objects and minimize collisions with movable objects. Since the locations of objects are not precisely known, the choices of poses and robot configurations take into account this uncertainty by avoiding, when possible, moving too near uncertain objects.

An example Consider the environment in figure 1 where uncertainty is shown by a translucent "shadow" centered at the mean of object poses and whose boundaries extend out two standard deviations. The goal is to place the red object (A) currently on the table onto the shelf, currently occupied by the green object (B) and the blue object (C). There isn't enough free space on the shelf for the robot to reach in with A in its hand, since the robot's hand is very wide. Also A cannot be grasped from above without colliding with the shelves and cannot be grasped from the front without colliding with the cyan object (D) on the table.

To simplify the presentation, we start out with relatively low uncertainty on the location of the objects; this is evident in figure 1 since the shadows are relatively small. The planner constructs a 17-step plan (inferential operations elided for space) to achieve the goal.

1 : Move(C(0.00, 0.00), C(0.59, -0.10))
2 : Pick(objC, left, 0, P(1.30, 0.10, 1.17))
3 : Move(C(0.59, -0.10), C(0.45, -0.60))
4 : Place(objC, left, 4, P(1.16, -0.30, 0.68))
5 : Move(C(0.453, -0.60), C(0.49, 0.47))
6 : Pick(objD, left, 0, P(1.10, 0.47, 0.68))
7 : Move(C(0.49, 0.47), C(0.50, 0.09))
8 : Place(objD, left, 4, P(1.10, 0.09, 0.68))
9 : Move(C(0.50, 0.09), C(0.51, 0.22))
10 : LookAt(objA, P(1.15, 0.35, 0.68))
11 : MoveNB((0.51, 0.22))
12 : Pick(objA, right, o, P(1.15, 0.35, 0.68))
13 : Move(C(0.51, 0.22), C(0.58, 0.36))
14 : Place(objA, right, 4, P(1.32, 0.19, 1.17))
15 : Move(C(0.58, 0.36), C(0.00, 0.00))
16 : LookAt(objA, P(1.32, 0.19, 1.17))
17 : AchIn(objA, shelf2)

For compactness, we denote the robot configuration as $C(x, y)$, representing the position of the base center and we denote object poses as $P(x, y, z)$. In actuality, 21-dof robot configurations are used; object poses are represented by an integer denoting a support face and a 4-dof (x, y, z, θ) pose.

The planning starts with the goal condition:

x* Bd[In[objA, shelf2], 0.950]

An $\star$ before a fluent indicates that it is not true in the current belief state; an x indicates that it is chosen for regression. This fluent is regressed (by action 17) to a goal of placing the object at a pose chosen so that it is within the shelf region; the choice of pose is subject to backtracking. The commitment to the object pose is dependent on a commitment on the pose distribution of the shelves.

B[Pose[shelves], P(1.35, 0.03, 0.68), 0.005, 0.01, 0.97]
x* B[Pose[objA], P(1.32, 0.19, 1.17), 0.005, 0.01, 0.97]

At this point, instances of *Place*, *Push*, or *Look* can achieve a *Pose* distribution; *Place* and *Push* affect the mean and *LookAt* affect the variance. The variance on the pose of A that is required to achieve In is too tight to be achieved by a *Place* or *Push* and so a *LookAt* is chosen. This means that the last action in this plan is a sensing operation that verifies the placement succeeded. If during execution, the result of the look shows that the placement did not succeed, then replanning happens.

Regressing the chosen fluent via action 16 , we get a new pre-image that requires the robot to be at (with a tolerance of 0.001) a configuration where the object is visible, in this case, the choice of configuration is the robot's configuration in the start state, but with the head pointed towards the object. We represent all uncertainties relative to the base of the robot and we assume that the proprioception error on the robot joints is negligible. Thus, the *Conf* fluent is a primitive fluent.

B[Pose[shelves], P(1.35, 0.03, 0.68), 0.005, 0.01, 0.97]
Bd[CanSeeFrom[C(0.00, 0.00), objA, []], 0.90]
* B[Pose[objA], P(1.32, 0.19, 1.17), 0.03, 0.01, 0.77]
x* Conf[C(0.00, 0.00), 0.001]

Importantly, the distribution on the pose of A in this pre-image is much broader (variance is 0.03 instead of 0.005); this provides a feasible target for a later placing operation.

We next choose to achieve the robot configuration; this is achieved with a *Move* (action 15). One precondition of this action is that the target configuration be reachable via a collision-free path from the (arbitrarily chosen) "home" configuration (*CanReachHome* is true). By requiring that all motions be able to connect to a home configuration, we simplify the planning, at some small loss of generality. Another precondition for this action is that the robot be at the starting configuration, however, we do not commit to a particular configuration at this point, since it is completely unconstrained. The value will be filled in after a more specific operation is chosen.

B[Pose[shelves], P(1.35, 0.03, 0.68), 0.005, 0.01, 0.97]
Bd[CanSeeFrom[C(0.00, 0.00), objA, []], 0.90]
x* B[Pose[objA], P(1.32, 0.19, 1.17), 0.03, 0.01, 0.77]
Bd[CanReachHome[C(0.00, 0.00), []], 0.80]
Conf[?C, 0.001]

Now, we tackle the object pose via a *Place* operation (action 14). This achieves the desired pose given that the object is grasped appropriately, the robot is in a suitable starting configuration and a suitable set of paths (approach with no object, depart with object) exist. The *CanPlace* fluent is a test on that relatively complex set of conditions.

B[Pose[shelves],(1.35, 0.03, 0.68), 0.005, 0.01, 0.97]
Bd[CanSeeFrom[C(0.00, 0.00), objA, [COND]], 0.90]
Bd[CanReachHome[C(0.00, 0.00), [COND]], 0.80]
* B[Grasp[objA, right, 0], 0.005, 0.009, 0.79]
* Bd[Holding[right], objA, 0.79]
* Bd[CanPlace[objA, right, P(1.32, 0.19, 1.17), []], 0.90]
* Conf[C(0.58, 0.36), 0.001]

In this pre-image, the pre-existing conditional fluents *CanSeeFrom* and *CanReachHome* are now conditioned with the resulting fluents from the *Place* operator. [COND] stands for:

[B[Pose[objA], P(1.32, 0.19, 1.17), 0.03, 0.01, 0.77],
Bd[Holding[right], none, 0.79]]

That is, the robot must be able to reach home and see A in a belief state that is like the starting belief, but modified so that the distribution for A's pose is as specified in the condition and the hand is empty.

The regression process proceeds until a pre-image is found where every fluent it satisfied by the starting belief.

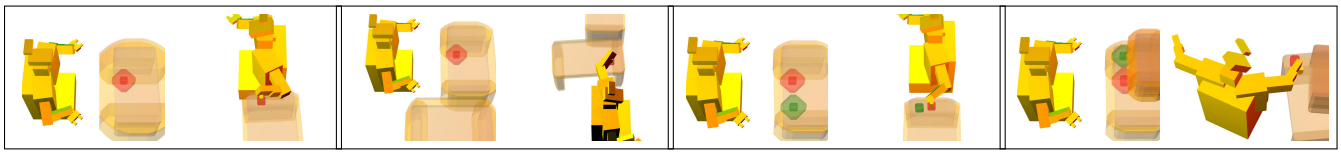


Fig. 3: Left: Initial state; Right: sample final state.

VII. Experiments and Conclusions

We tested the planner in 4 simulated scenarios, see figure 3; the size of the "shadows" reflect the uncertainty in poses. The examples involve: (1) place an object at the left end of the table, (2) place an object anywhere on another table, (3) place two objects anywhere at the left end of the table, and (4) push a block, too large to grasp, to the left end of the table; this requires moving a blocking object out of the way. The simulation includes simple models for error in each of the actions, so that an open-loop plan is unlikely to succeed. We evaluated four settings for the planner. **Hier/Flat** indicates whether the planner was hierarchical or not. **Monitor/Replan** indicates whether we monitor and replan as needed or we replan after every action. The running times (in seconds) are shown in the table below; they represent averages over 5 executions. Each execution involves a random draw from the initial pose distribution for each of the object poses. The empty entries exceeded the maximum allowed running times of 1000 secs.

Test	Hier/Mon	Hier/Replan	Flat/Mon	Flat/Replan
1	87	152	198	377
2	132	249	389	718
3	146	440	—	—
4	172	552	—	—

These results support the fact that (a) hierarchical planning is effective in these types of problems and (b) that exploiting pre-images to replan only when necessary provides a substantial time savings. In the flat setting, for tests 1 and 2, constructing the first complete plan takes 84 secs. That is nearly the full running and execution time in hierarchical mode. For continual replanning, test 1 and 2 construct an average of 10.2 and 14.5 (flat) plans per trial respectively, corresponding to the average length of the plans. When replanning only as needed in tests 1 and 2, 4.2 and 7.0 (flat) plans are constructed on average. The time to create subsequent plans drops quickly, both because the length of the plans decrease and due to geometric caching.

The construction of explicit pre-images enables hierarchical planning and execution monitoring, thus supporting a "determinize and re-plan" style of planning under uncertainty and enabling teh solution of complex manipulation planning problems with real uncertainty.

References

[1] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *IJRR*, vol. 32, no. 9-10, 2013.

[2] T. Lozano-Pérez, M. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *IJRR*, vol. 3, no. 1, 1984.

[3] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning: Theory and practice*. Elsevier, 2004.

[4] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[5] J. F. Canny and J. H. Reif, "New lower bound techniques for robot motion planning problems," in *FOCS*, 1987.

[6] M. Levihn, L. P. Kaelbling, T. Lozano-Pérez, and M. Stilman, "Foresight and reconsideration in hierarchical planning and execution," in *IROS*, 2013.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, 1998.

[8] D. S. Weld, "Recent advances in AI planning," *AI Magazine*, 1999.

[9] R. Platt, R. Tedrake, L. P. Kaelbling, and T. Lozano-Pérez, "Belief space planning assuming max. likelihood observations," in *RSS*, 2010.

[10] T. Erez and W. Smart, "A scalable method for solving high-dimen. continuous POMDPs using local approximation," in *UAI*, 2010.

[11] N. E. du Toit and J. W. Burdick, "Robotic motion planning in dynamic, cluttered, uncertain environments," in *ICRA*, 2010.

[12] S. W. Yoon, A. Fern, and R. Givan, "FF-Replan: A baseline for probabilistic planning," in *ICAPS*, 2007.

[13] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, 2005.

[14] R. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, no. 1-3, 1972.

[15] C. Fritz and S. A. McIlraith, "Monitoring plan optimality during execution," in *ICAPS*, 2007.

[16] J. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge U. Press, 2006.

[18] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with prob. roadmaps," *IJRR*, vol. 23, no. 7–8, 2004.

[19] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manip. task," *IJRR*, vol. 30, 2011.

[20] J. Barry, L. P. Kaelbling, and T. Lozano-Perez, "A hierarchical approach to manipulation with diverse actions," in *ICRA*, 2013.

[21] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *ICAPS*, 2009.

[22] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning for robotic manipulation," in *ICRA*, 2011.

[23] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *IROS*, 2012.

[24] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, 2009.

[25] E. Plaku and G. Hager, "Sampling-based motion planning with symbolic, geometric, and differential constraints," in *ICRA*, 2010.

[26] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *ICRA*, 2014.

[27] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *WAFR*, 2014.

[28] ——, "Backward-forward search for manipulation planning," in *IROS*, 2015.

[29] M. Stilman and J. J. Kuffner, "Planning among movable obstacles with artificial constraints," in *WAFR*, 2006.

[30] K. K. Hauser, "The minimum constraint removal problem with three robotics applications," *I. J. Robotic Res.*, vol. 33, no. 1, pp. 5–17, 2014. [Online]. Available: http://dx.doi.org/10.1177/0278364913507795