2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Unobservable Monte Carlo Planning for Nonprehensile Rearrangement Tasks

Jennifer E. King, Vinitha Ranganeni, Siddhartha S. Srinivasa
The Robotics Institute, Carnegie Mellon University
{jeking, vrangane, ss5}@andrew.cmu.edu

***Abstract*— In this work, we present an anytime planner for creating open-loop trajectories that solve rearrangement planning problems under uncertainty using nonprehensile manipulation. We first extend the Monte Carlo Tree Search algorithm to the unobservable domain. We then propose two default policies that allow us to quickly determine the potential to achieve the goal while accounting for the contact that is critical to rearrangement planning. The first policy uses a learned model generated from a set of user demonstrations. This model can be quickly queried for a sequence of actions that attempts to create contact with objects and achieve the goal. The second policy uses a heuristically guided planner in a subspace of the full state space. Using these *goal informed* policies, we are able to find initial solutions to the problem quickly, then continuously refine the solutions as time allows. We demonstrate our algorithm on a 7 degree-of-freedom manipulator moving objects on a table.**

I. Introduction

In this work we generate open-loop trajectories that solve the rearrangement planning problem [1]–[5] using nonprehensile manipulation. In these problems, a robot must plan in a cluttered environment, reasoning about moving multiple objects in order to achieve a goal.

Nonprehensile interactions have proven to be a powerful strategy for pregrasp manipulation [6]–[8], manipulating large or heavy objects [1] and manipulating in clutter [9]–[12]. However, open-loop execution of trajectories that incorporate nonprehensile actions are prone to failure due to uncertainties in object and robot pose and in the physical modeling of the interactions.

Prior work has shown that nonprehensile interactions such as the push-grasp [1], [13] can be inherently uncertainty reducing and successfully executed open-loop if the uncertainties in pose and interaction are considered when generating the motion. This analysis of pushing under uncertainty has been limited to short simple motions such as a straight line push. In this work we consider the following question: *How do we generalize this analysis to create robust open-loop pushing trajectories for use in rearrangement tasks?*

This generalization to full rearrangement trajectories introduces three inherent challenges. First, rearrangement planning occurs in a continuous high dimensional state space that describes the state of the robot and movable objects. This requires our planner search across an infinite dimensional belief space in order to account for state uncertainties.

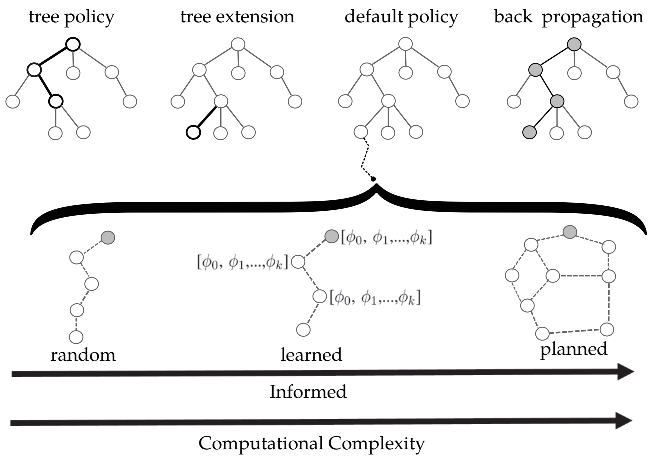

Fig. 1. Unobservable Monte Carlo Planning based on Monte Carlo Tree Search. The planner relies on a *default policy* to compute the potential for goal achievement from a node in the tree. We propose two policies: learned and planned. These policies trade-off additional computational complexity for better informed decision making that improves the efficiency of tree growth.

Second, contact between robot and objects causes physics to evolve in complex, non-linear ways and quickly leads to multimodal and non-smooth distributions. This makes methods that rely on closed form representation of the stochastic dynamics inapplicable [14].

Third, most actions in the continuous action space fail to make and, importantly, sustain meaningful contact with objects. Unlike grasping, the motion of a pushed object cannot be modeled as rigidly attached to the robot. Instead, the motion is directly governed by the physics of the interaction between robot and object. Once contact is made, only a small subset of the continuous action space will sustain this contact and move the object.

In this work, we propose an Unobservable Monte Carlo Planner (UMCP) that extends Monte Carlo Tree Search (MCTS) methods into unobservable domains. MCTS methods naturally deal with the first two challenges. The algorithm focuses the search to beliefs reachable from a known initial belief state and uses Monte Carlo simulations to approximate the unknown stochastic dynamics.

Central to MCTS algorithms is the use of a *default policy* to guide a simulation, or rollout, that quickly evaluates the potential value from a state. The simplest default policies perform random rollouts, drawing action sequences from a

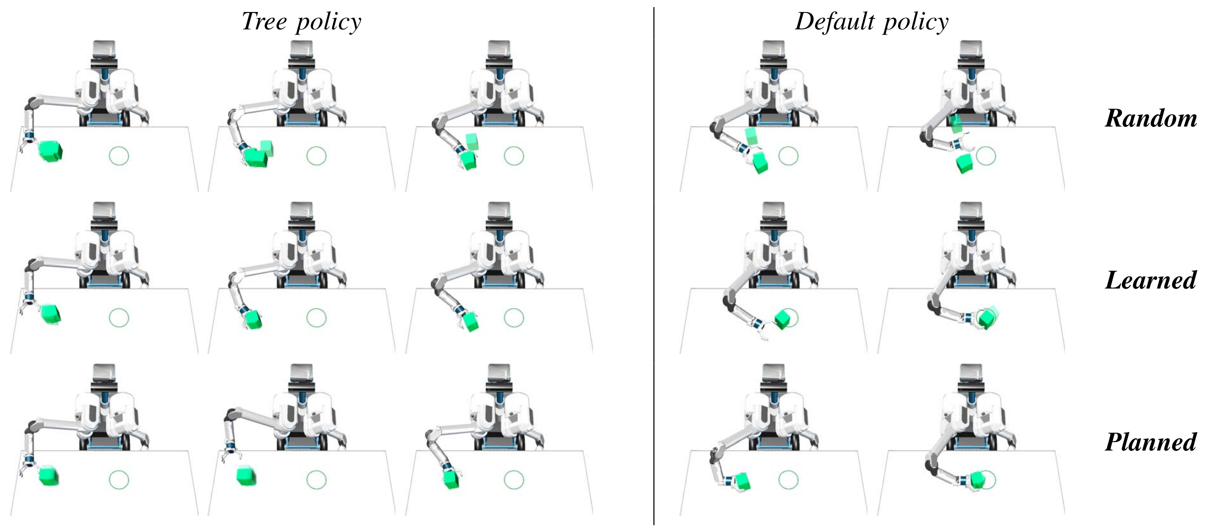

Fig. 2. An example path computed with the *random* default policy, *learned* default policy and *planned* default policy after $t = 45$ s of planning time. The portion of the paths extracted using the tree policy look similar. However, the *random* policy quickly breaks contact with objects and is unable to move any portion of the belief to the goal. The *learned* and *planned* policies are able to use knowledge of the problem to maintain contact and achieve the goal.

distribution over the control space defined for the robot. In rearrangement planning, random rollouts will rarely be informative, as most action sequences fail to make meaningful contact with objects. Relying on this default policy restricts the amount of information the planner can use to guide tree growth.

Our insight is that by carefully selecting a *goal informed default policy* that generates actions with the goals of rearrangement planning and nonprehensile interaction in mind, we can extract useful trajectories from the planner much earlier to better guide the search. We propose two such policies. The first uses user demonstrated trajectories to learn a mapping from state to control space. This mapping can then be used in place of random selection to generate rollouts. The second uses a planner that can solve rearrangement planning in the lower dimensional subspace containing only objects critical for goal achievement. The sequence of actions generated by the planner is then used to perform a rollout in the full state space.

These policies trade-off addition computational complexity for more informed decision making to guide tree growth (Fig.1). We test both policies on a manipulation task requiring a robot to push objects on a table. Our results show the *learned default policy* is useful when working in low clutter (Fig.2) while the *planned default policy* better guides the search through high clutter.

The remainder of this paper is structured in the following way. In Sec.II we formalize the rearrangement planning problem. In Sec.III we outline the UMCP planner and our proposed default policies. We demonstrate the effectiveness of the algorithm in Sec.V. Finally, we discuss limitations and areas for future work in Sec.VI.

II. The Rearrangement Planning Problem

A. Terminology

Our environment contains a robot $\mathcal{R}$, a set, $\mathcal{M}$, of objects that the robot is allowed to manipulate and a set, $\mathcal{O}$, of obstacles which the robot is forbidden to contact. We define the state space of the planner X as the Cartesian product of the state spaces of the robot and all objects in $\mathcal{M}$: $X = X^{\mathcal{R}} \times X^1 \times \cdots \times X^m$. We define the free state space $X_{free} \subseteq X$ as the set of states where the robot and objects are not contacting the obstacles and are not penetrating themselves or each other. Note that this definition specifically allows contact between robot and movable objects, which is critical for manipulation.

We consider pushing interactions. Thus, the motion of the movable objects is governed by the physics of the environment and the contact between the objects and the robot. As a result, the state x evolves non-linearly based on the physics of the manipulation. We describe this evolution as a non-holonomic constraint:

$$\dot{x} = f(x, u) \tag{1}$$

where $u \in \mathcal{U}$ is an instantaneous control input to the robot.

The task of rearrangement planning is to find a *feasible* trajectory $\xi : \mathbb{R}^{\geq 0} \rightarrow X_{free}$ from an initial state $x_0 \in X_{free}$ to any state in a goal region $X_{\text{G}} \subseteq X_{free}$. A path is feasible if there exists a mapping $\pi : \mathbb{R}^{\geq 0} \rightarrow \mathcal{U}$ such that Eq.(1) holds throughout the duration T of the trajectory: $\dot{\xi}(t) = f(\xi(t), \pi(t))$ for all $t = [0, \ldots, T]$.

B. Uncertainty

We wish to generate open-loop plans robust to the uncertainties prevalent when executing trajectories in the real world. In general, open-loop plans are susceptible to failure due to uncertainty in initial state and poor modeling of both the motion of the manipulator and the physics of the interaction (Fig.3).

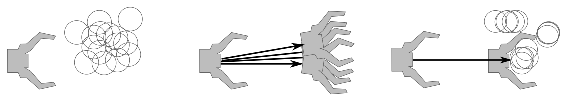
Fig. 3. Three sources of uncertainty in rearrangement planning with nonprehensile interaction: (*left*) Initial state. (*center*) Manipulator motion. (*right*) Physical interaction.

We represent initial state uncertainty as a belief $b_0 = p(x_0)$ that describes a probability distribution over possible initial states $x_0 \in X_{free}$. We represent modeling errors by assuming our state evolves as a stochastic non-holonomic system. This induces a distribution $p(\xi|\pi, x)$ over the trajectories that result from executing a control sequence π from a state x under the stochastic transition dynamics.

We represent the rearrangement problem as an instance of conformant probabilistic planning [15] where the goal is to maximize the probability that executing a sequence of actions π results in goal achievement. We can express this probability as an expectation:

$$p_\pi(x) = \int_\Xi \mathbf{1}_G(\xi) p(\xi|\pi, x) d\xi \tag{2}$$

where Ξ is the set of all trajectories from a state x and $\mathbf{1}_G : \xi \to \{0, 1\}$ is the indicator function that returns 1 if the endpoint of ξ is in X_G. Our goal is to generate a control sequence $\pi^* \in \Pi$ that maximizes the total probability of π^* achieving success (p_{π^*}) given all uncertainties:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}\, p_\pi \tag{3}$$

$$= \underset{\pi \in \Pi}{\operatorname{argmax}} \int_{x \in X} b_0(x) p_\pi(x) dx \tag{4}$$

In this work, we consider mappings π instantiated as a sequence of discrete actions $\pi = \{a_1, \dots, a_j\}$ where each action $a_i = (u_i, \Delta t)$ represents a control input to the robot and a duration to apply the control.

III. Monte Carlo Tree Search

In our domain, the evolution of the uncertainty when using nonprehensile interactions is non-smooth and non-Gaussian. These characteristics make closed form representation of the system dynamics difficult. As a result, exact computation of Eq.(2) is not possible.

Monte Carlo methods have been used widely when the exact dynamics are unknown or difficult to model [16]–[20]. These methods use a generative model, G, or black-box simulator, to sample successor states given a current state and an action: $x' \sim G(x, a)$.

Monte Carlo Tree Search [19], [21] (MCTS) is one such algorithm that uses this paradigm. The MCTS algorithm iteratively builds a tree using Monte Carlo simulations. The tree estimates the value of action sequences by tracking the mean reward obtained from simulations of the sequences.

MCTS is a good fit for our problem. We can use a physics model to perform the black-box simulations. These physics simulations have some computational expense. The MCTS framework efficiently focuses computational resources to relevant regions of state space. In addition, the algorithm is anytime and highly parallelizable.

Algorithm 1 Unobservable Monte Carlo Planning

```
1:  $s_0 \leftarrow$ GenerateInitialSamples()
2:  while not timeout do
3:      $x \leftarrow$ SampleState($s_0$)
4:      Simulate($x, \{\}, 0$)
5:  function Simulate($x$, $h$, $d$)
6:      if $\gamma^d < \epsilon$ then return 0
7:      if NotVisited($h$) then
8:          InitializeHistory($h$)
9:          return DefaultPolicy($x$)
10:     $a \leftarrow$ TreePolicy($h$)
11:     $x' \leftarrow G(x, a)$
12:     $r \leftarrow R(x, a) +$
13:         $\gamma \cdot$ Simulate($x'$, $h \cup \{a\}$, $d + 1$)
14:     $\hat{\mathcal{B}}(h) \leftarrow \hat{\mathcal{B}}(h) \cup \{x\}$
15:     $N(h) \leftarrow N(h) + 1$
16:     $\hat{Q}(h, a) \leftarrow \hat{Q}(h, a) + r$
17:     return $r$
```

IV. Unobservable Monte Carlo Planning (UMCP)

The POMCP [19] algorithm applies the MCTS framework to partially observable environments. We use a similar approach to plan in our unobservable environment. We build a tree such that each node represents a unique history, $h = \{a_1, \dots a_t\}$. Three values are stored for each node: $N(h)$ - the number of times the history, or action sequence, has been explored, $\hat{Q}(h, a)$ - an estimate of the value of taking action a after applying history h, and $\hat{\mathcal{B}}(h)$ - an estimate of the true belief achieved when applying the actions in h from known initial belief b_0.

Alg.1 shows the UMCP algorithm which applies MCTS to an UMDP. The tree is rooted with an initial belief state s_0 that contains a set of states drawn from an initial distribution defined on the state space. Then, during the search an initial state $x \sim s_0$ is drawn from the belief state (Line 3). This state is propagated through the tree by using the *tree policy* to select actions (Line 10) and using a noisy physics model to forward propagate the state under the selected actions (Line 11). After applying the physics model, the new state is added to the belief state of the history (Line 14). The search recurses through the tree, propagating a single state through the noisy transition dynamics. Over time, the belief states represented at the nodes of the tree grow to represent the true belief distribution.

Once the search reaches a previously unvisited history, a *default policy* is used to rollout the remainder of a simulation and accumulate reward (Line 9). This reward is propagated back through the tree to update the value function estimate stored for each history.

A. Reward model

As stated in Sec.II, our goal is to generate paths that maximize the probability of successful execution. We encode this goal in our reward model:

$$R(x,a) = \mathbf{1}_G(x) \tag{5}$$

Here the indicator function returns 1 if $x \in X_{\mathrm{G}}$.

B. Action set

MCTS-based planners search across a discrete action set. The naive method for generating a discrete action set from our continuous space $\mathcal{U}$ is to divide the space into partitions, or bins, and select a single representative control from each bin and create an action that applies this control for a fixed duration: $a = (u, \Delta t)$. These *basic* actions are context agnostic: they ignore the goal of the planning instance.

We know *contact* is critical for goal achievement in rearrangement problems. We augment the action set to account for this by generating specific *contact* actions aimed at maintaining contact with objects important to goal achievement.

These *contact* actions are state dependent and must be dynamically generated for each node, or history, in the tree. We instantiate *contact* actions using the first state in the estimated belief $x \in \hat{B}(h)$ for each history. A *contact* action is generated by solving the two-point BVP in the robot's state space that moves the robot to a pose in contact with an object based on the object's pose in x. We create one *contact* action for each object in x defined in the goal.

The result is a discrete action set $\mathcal{A}_h = \mathcal{A}_{basic} \cup \mathcal{A}_{cont}$ for each history h composed of a set $\mathcal{A}_{basic}$ of *basic* actions that move the robot without the explicit intent of creating contact with objects and a set $\mathcal{A}_{cont}$ of *contact* actions that explicitly contact important objects in the scene.

C. Tree Policy

The tree policy is used to select actions, or edges, in the UMCP tree to traverse. On the first visit to a given node in the tree corresponding to history h, the method from the previous section is used to generate a discrete set of actions $\mathcal{A}_h$. On subsequent visits, we follow the UCT algorithm [17] and use UCB1 [22] to select a single action from this set to traverse as follows:

$$a_t = \underset{a \in \mathcal{A}_h}{\operatorname{argmax}} \frac{\hat{Q}(h,a)}{N(h \cup \{a\})} + c\sqrt{\frac{\log N(h)}{N(h \cup \{a\})}} \tag{6}$$

where $c > 0$ is an exploration constant. Note that this selection method requires all actions are tried at least once.

The use of such a method is ideal because it provides a formal method for trading between exploration and exploitation.

D. Default policy

Each time the search reaches a leaf in the tree, the default policy is used to estimate the reward that will be obtained if we follow a path that leads through this leaf. The most common default policy is to randomly select a sequence of actions to apply. For our rearrangement planning problem, this policy will rarely be informative: most action sequences fail to create and maintain the contact with objects that is critical to goal achievement.

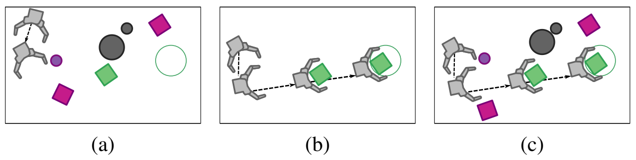

Fig. 4. An example of MCTS applied to rearrangement planning using the planned default policy. (a) A *tree policy* is used to select initial actions. (b) The *default policy* plans in the lower dimensional space containing only objects important to goal achievement. (c) The resulting path is propagated through the full space to generate a reward.

Instead, we define two informed default policies that are capable of evaluating the potential to achieve the goal. The first uses a model learned from user demonstrated trajectories to generate a rollout. The second uses a heuristically guided planner in a subspace of the full state space. In the following sections we outline these policies.

Learned default policy

Our intuition is that we can use human judgment to learn a good default policy. Our goal is to learn a mapping: $\rho : X \to \mathcal{U}$ from a state $x \in X$ to a control $u \in \mathcal{U}$ from a set of human demonstrations. We can use any method for collecting these demonstrations and learning the mapping. We discuss a specific choice in the experiments.

Once we have learned a mapping ρ, we can use it as a default policy by creating a fixed length rollout. To do this we repeatedly use ρ to generate a control and G to forward propagate the control for a fixed duration Δt:

$$x_{t+1} = G(x_t, (\rho(x_t), \Delta t))$$

We compute the reward achieved by the rollout using the final state of the rollout.

Planned default policy

The learned default policy attempts to generalize demonstrations to solve an instance of rearrangement planning. In this section we explore an alternative approach: we apply a state space planner to quickly search for a solution to the specific rearrangement problem.

We perform the search in the lower dimensional subspace containing only the elements of the full state space that are defined in the goal.

After generating a sequence of actions that solve the problem in the lower dimensional subspace, the actions are then forward simulated through the full state space using G and reward is calculated from the final reached state. Fig.4 illustrates this method.

By reducing the dimensionality of the state space in the default policy search, we allow for the possibility of using fast planners or exact solvers that provide much more information than random action sequences. We discuss a specific example in the experiments.

E. Path extraction

We use our tree to create an anytime algorithm for extracting paths. Upon request, a path π is extracted from the tree as follows. First, we extract π_{tree} by repeatedly picking the action a_t such that:

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}_h} \hat{Q}(h, a)$$

Once a leaf is encountered, we query the belief represented by the history $\hat{\mathcal{B}}(h)$ to obtain an estimated probability of success $\hat{p}_{\pi_{tree}} = \sum_{x \in \hat{\mathcal{B}}(h)} \mathbf{1}_G(x)$. If this probability is lower than the estimated probability of success of the best path found so far, $\hat{p}_{\pi^*}$, we randomly select a state from the belief $x \in \hat{\mathcal{B}}(h)$ and use the *default policy* to generate a path π_{def} from x to the goal. If successful, all remaining samples in $\hat{\mathcal{B}}(h)$ are forward propagated through this path to get an updated probability of success $\hat{p}_\pi$ of the combined path π formed from appending π_{def} to π_{tree}. If $\hat{p}_\pi > \hat{p}_{\pi^*}$ the path is returned. Otherwise, the previous best path π^* is returned.

The use of a *goal informed default policy* means we can often find path segments π_{def} that achieve the goal with non-zero probability. Our insight is that these path segments can be particularly useful when there is not enough planning time to deeply grow the tree, i.e. $\hat{p}_{\pi_{tree}} = 0$.

V. Experiments and Results

To test the capabilities of the UMCP algorithm, we task our robot HERB [23] with pushing a box on a table to a goal region of radius 0.1 m using the 7-DOF right arm. We test three hypotheses:

H.1 Using explicit **contact** actions allows the UMCP planner to generate paths with higher probability than a planner that uses a **basic** action set formed by discretization of each dimension of the control space.

H.2 The UMCP planner using the *goal informed* **learned** or **planned** default policy generates paths with higher probability of success than the UMCP planner with a **random** default policy.

H.3 The UMCP planner that uses **contact** actions and the *goal informed* default policy is able to produce paths that exhibit higher probability of success compared to baseline state space planners.

H.1 tests the need for actions that explicitly try to create contact with goal critical objects. **H.2** verifies our intuition that using a *goal informed* default policy will guide tree growth better than a random default policy. Finally, **H.3** verifies the need to track the evolution of uncertainty through sequences of actions during planning.

In the following sections we detail our planning setup and provide results for each hypothesis.

A. Test setup

We run each version of the UMCP planner 50 times in simulation on a scene with only one movable object (denoted low clutter in all results) and a scene with six movable objects (denoted high clutter in all results). The scenes are depicted in Fig.6-*top*. In each scene, we generate

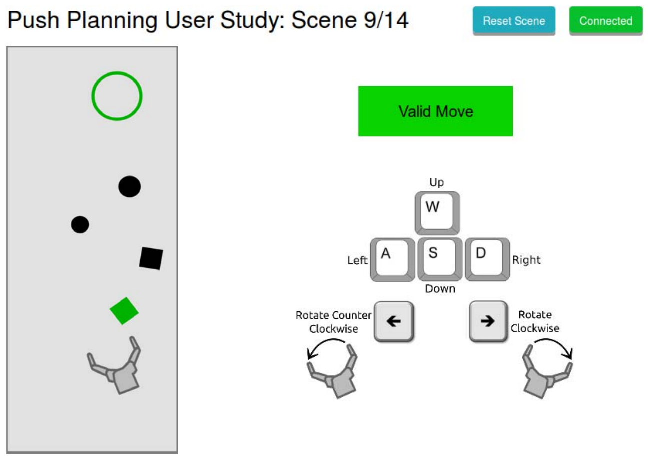

Fig. 5. An example interface used to collect user demonstrated rearrangement trajectories

the initial belief s_0 by sampling noise into the initial pose of each object from a Gaussian with distribution $\mu = \mathbf{0}, \Sigma^{1/2} = \mathtt{diag}\{2\mathtt{cm}, 2\mathtt{cm}, 0.1\mathtt{rad}\}$. This distribution was selected to reflect the actual distribution of noise from the object detection system used on HERB. We allow the UMCP planner to run for 300 s. We request and record a path every 15 s.

Following [12] we constrain the end-effector to move in the xy-plane parallel to the table in order to create motions likely to pushing objects. This allows us to define our control space $\mathcal{U}$ as the set of feasible velocities for the end-effector. We convert these end-effector velocities to full arm velocities using the Jacobian psuedo-inverse:

$$\dot{q} = J^{\dagger}(q)u + h(q)$$

where q is the current arm configuration, u is the end-effector velocity and $h : \mathbb{R}^7 \to \mathbb{R}^7$ is a function that samples the nullspace.

B. Learned default policy

We collect workspace trajectories from 97 users using Amazon Mechananical Turk. To collect these trajectories, we generate a set of seven scenes that require a robot to push an object on a table to a goal region. Each user is asked to solve each scene twice, for a total of 14 demonstrations from each user. For each demonstration, we record and save the sequence of state/action pairs, (x_t, a_t) used to solve the scene.

We provide users with an interface that allows them to apply discrete actions using keyboard input. Fig.5 shows an example interface. To simplify the user task, we render only the end-effector of the robot. In the example shown, the user must guide the robot hand to push the green box into the green circle. The action corresponding to the keyboard input from the user is pushed through a physics model to generate an updated pose of the robot and objects.

We identify and extract a set of relevant features Φ of a state $x \in X$. Example features are the number of objects blocking a direct path to the goal or the distance to the goal. Then, for each of the 112,928 state/action pairs (x_i, a_i)

collected from users, we create feature vectors ϕ_i from state x_i labeled by the action a_i to use as training data for a multi-class classifier. We train a random forest classifier using the scikit-learn python package [24]. This classifier serves as the mapping ρ needed to perform rollouts.

C. *Planned default policy*

We use a weighted A* search [25] with $w = 5.0$ for our planned default policy. The search uses the same discrete action set used to build the UMCP tree, including the *contact* actions critical to achieving success.

The search runs in the lower dimensional subspace containing only the robot and the box the robot must push to the goal. To guide the search, we compute the cost of an action a as the distance in workspace the robot moves during the action. Then we define an admissible heuristic, $h(x) = d_{cont}(x) + d_{move}(x)$, where d_{cont} is the minimum workspace distance the robot must move to make contact with the box and d_{move} is the minimum workspace distance the box must move to achieve the goal.

The planner is allowed 1 s to search. The result of the search is a sequence of actions $\{a_1, \ldots, a_k\}$ that describe robot motions to achieve the goal in the lower dimensional subspace. The action sequence is then applied in the full state space to check for validity and compute reward.

D. *Baseline planners*

We compare the estimated success rate of the paths generated by the UMCP algorithm to an anytime version of the B-RRT from [26]. Briefly, the B-RRT planner generates rearrangement plans by using a Rapidly-exploring Random Tree (RRT) [27] to search through state space. During tree growth, each action is evaluated for its robustness to uncertainties and this evaluation is used to bias tree growth–robust actions are more likely to be selected for extensions. Importantly, the planner does not track uncertainty through sequences of actions. Instead it makes local decisions about the robustness of each action. We test against two versions of the algorithm. First, we set $b = 0.0$ to eliminate all bias. This reduces the planner to a search over state space that does not consider uncertainty. Second, we set $b = 2.0$. This biases the search to prefer uncertainty reducing actions.

To create an anytime version of the planner, we make as many repeated calls to the planner as possible within 300 s. When a call completes we perform a set of 100 noisy rollouts on the resulting control sequence π to generate an estimate $\hat{p}_\pi$ of the probability of success. We generate these noisy rollouts using the same noise parameters used to create the initial belief s_0 in the UMCP planner. We keep π only if it has higher estimated success probability than all previous paths generated by the planner. This is a similar algorithm to the AMD-RRT described in [26] though we use probability of success rather than performance of the path under a divergence metric.

E. *Effect of contact actions*

We compare the UMCP planner with **contact** actions to the UMCP planner with only the **basic** actions formed from discretization of the control space. Both version of the planner use the planned default policy.

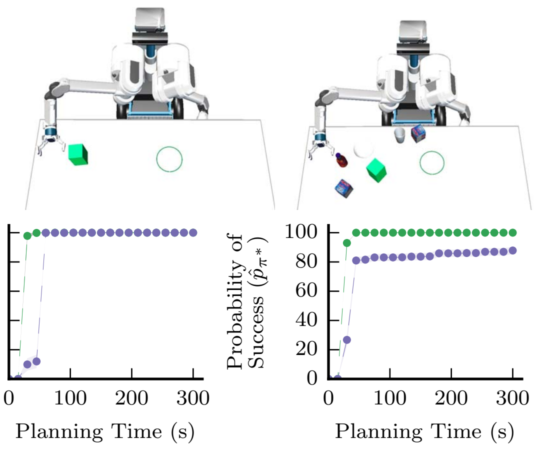

Fig. 6. Using **contact** actions (—) allows for finding better paths sooner than using only **basic** action (—) on both a low clutter scene (*left*) and a high clutter scene (*right*).

Fig.6 compares the probability of success $\hat{p}_{\pi*}$ of the path returned by the planner at each time step. In low clutter scenes, the usefulness of the **contact** actions is limited (Fig.6-*left*). The UMCP planner with the **contact** actions finds paths only slightly faster. This is due to the use of the **planned** default policy to complete paths extracted from the tree. Examination of the generated paths shows that the default policy is heavily relied upon to generate the contact needed for success when using only the **basic** action set.

The benefit of these actions is much more prevalent in high clutter scenes (Fig.6-*right*). Here, the **planned** default policy fails to be applied without first moving either the bowl or bottle. The **basic** action set is not rich enough to create useful contact. In contrast, the **contact** action easily moves both objects out of the way in order to make contact with the box. Then the **planned** default policy can be applied to achieve the goal. Fig.9 depicts an example path found by the UMCP planner with **contact** actions.

These results support **H.1**: Using explicit contact actions allows the planner to generate paths with higher probability than a planner that uses a basic action set formed by discretization of each dimension of the control space.

F. *Effect of default policy*

1) Low clutter scene: Next we examine the effect of our choice of default policy. Fig.7-*left* shows the estimated probability of success of the path output by the UMCP planner, $\hat{p}_{\pi^*}$, as a function of planning time for a low clutter scene. As can be seen, the use of the **planned** default policy allows us to generate path with high probability of success faster than the planner with the random default policy. This is despite the planned default policy taking almost 10x longer to compute than the random default policy (mean time 0.5 s and 0.06 s respectively). The **learned** default policy also outperforms the random default policy on this scene, finding solutions nearly as good as the **planned** policy.

Perhaps more interesting are the qualitative aspects of the results. Fig.2 shows an example path at $t = 45$ s for the

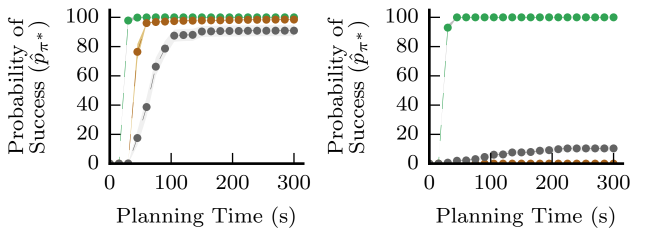

Fig. 7. Use of *goal informed* default policies such as the planned (—) and learned (—) result in overall better paths when compared to using a policy that randomly selects actions (—) for both low clutter (*left*) scenes and high clutter scenes (*right*).

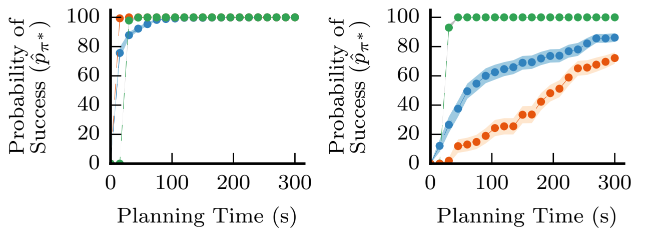

Fig. 8. On simple scenes (*left*), the B-RRT with bias $b = 2.0$ (—) is able to find better paths quickly. The UMCP algorithm (—) is able to find better paths more quickly than the B-RRT with no bias ($b = 0.0$) (—). On difficult scenes (*right*), the UMCP outperforms both instances of the B-RRT.

UMCP planner with each version of the default policy. The left column shows the portion of the path π_{tree} extracted from the tree. The right column shows the portion of the path π_{def} extracted using the default policy.

The tree paths for the **planned** and **random** default policies are similar, though the UMCP planner that uses the **planned** default policy finds a more robust sequence (Fig.2-*top*). This is because the default policy is more informative and allows better estimates of $\hat{Q}(h, a)$ early in the tree.

The main difference in the two results comes from the portion of the path extracted using the default policy. The **planned** default policy maintains contact with the goal object and eventually moves the full belief either into or near the goal region. The **random** default policy loses contact with the object quickly and fails to move any of the belief to the goal. Interestingly, the **learned** default policy is fast and informative enough that the tree is able to grow almost all the way to the goal within the 45 s.

2) High clutter scene: Fig.7-*right* shows results for a high clutter scene. Again, the **planned** default policy performs well, returning paths with higher probability of success than the planner using the **random** policy. Interestingly, the **learned** default policy struggles to find solutions to this scene and performs worse than the **random** policy. This scene is difficult. The bottle blocks direct access of the box. It must be moved or avoided. The random policy fails to find any valid sequences of actions that achieve the goal for most of the search. As a result, no reward is propagated through the tree, and the tree fails to grow deep. Similarly, the **learned** policy fails to find good solutions, resulting again in a tree with almost no reward. Conversely, the **planned** policy is able to explicitly search for and find valid solutions for *this* problem. As a result, the tree contains sufficient reward to properly guide the search and find robust paths. Fig.9 depicts a solution found by the planner.common failure by

Our results partially support **H.2**: The *goal informed* **planned** default policy is useful in low and high-clutter scenes.

G. Comparison to baseline planners

Finally, we compare the UMCP planner using **contact** actions in the action set and the **planned** rollout policy to the baseline planners described in Sec.V-D. Fig.8 shows the estimated probability of success $\hat{p}_{\pi^*}$ of each planner as a function of planning time.

For the low clutter scene both versions of the B-RRT are easily able to find solutions (Fig.8-*left*). The B-RRT with $b = 2.0$ performs exceptionally well here because there exists a solution comprised almost entirely of uncertainty reducing, or low divergence, actions. These solutions are also found by the UMCP planner (Fig.2-*top*).

The advantage of the UMCP algorithm can be seen for the high clutter scene (Fig.8-*right*). Here, the B-RRT performs poorly because the actions that reduce uncertainty in the box increase uncertainty in the pose of other objects, such as the bottle or bowl (Fig.9). Such actions perform poorly under the divergence metrics used to bias the B-RRT. As a result, the B-RRT is slow to explore and find solutions. The UMCP allows for increasing uncertainty along dimensions that are not important for goal achievement. This allows the planner to find solutions more easily.

Overall, for planning time budgets greater than 30 s the UMCP algorithm finds solutions as good as the solutions found by the B-RRT algorithms in the low clutter scene. The UMCP algorithm outperforms both baseline planners in the high clutter scene. This supports **H.3**: Our UMCP planner that uses contact actions and the planned default policy is able to produce paths that exhibit higher probability of success compared to anytime versions of baseline planners.

VI. Discussion

In this work we propose an Unobservable Monte Carlo Planner. This algorithm extends MCTS to the unobservable domain. We show that by carefully selecting an informative default policy and an action set capable of generating contact with important objects in the scene, we are able to plan solutions that are robust to uncertainty. The result is an anytime algorithm that quickly returns good solutions in the example scenarios.

Our experiments show that the *planned* default policy performed exceptionally well on the rearrangement tasks we consider. We concede that this task is particularly well suited for the planned default policy: only a single movable object is described in the goal, making the reduced state space containing only this object easy to search using a heuristic planner. More complicated rearrangement tasks that contain several objects defined in the goal, such as those in [28], may benefit less from this approach as the lower dimensional planning problem will be quite difficult to solve. We believe the *learned* default policy, trained with a sufficiently large and representative data set, may be more effective in these

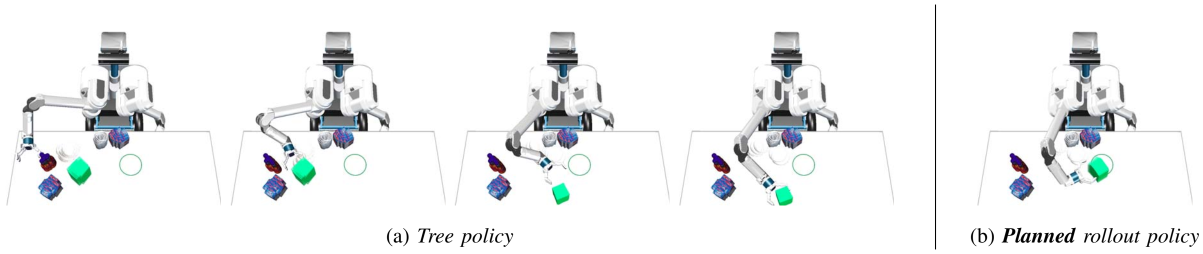

Fig. 9. In high clutter scenes, the UMCP algorithm with **contact** actions performs well. The algorithm allows for increasing uncertainty in the pose of the bowl, as long as it does not inhibit goal achievement.

tasks. To generate such a data set, we can supplement the human demonstrations with synthetic demonstrations generated from random simulations.

This algorithm represents a step toward planning rearrangement tasks using nonprehensile manipulation in belief space. We believe there are three promising directions that may improve the quality of the planner. First, we can improve the reward function. The binary nature of our current reward function makes it difficult to evaluate incremental progress during the search. Including additional information such as contact between manipulator and objects may allow for identifying promising search directions earlier. Second, we can expand the set of actions considered by the planner by using gradient free methods to make local adjustments to the action set [29]. Finally, we believe this algorithm could be extended to closed-loop planning by incorporating feedback as observations in a full POMDP formulation. Careful thought must be applied to allow us to maintain tractability under the exponential increase in histories due to the introduction of observations. However, recent work in using contact sensing [30] during push-grasping shows this may be possible.

References

[1] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," in *Robotics: Science and Systems*, 2011.

[2] D. Nieuwenhuisen, A. Stappen., and M. Overmars, "An effective framework for path planning amidst movable obstacles," in *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[3] J. Scholz and M. Stilman, "Combining motion planning and optimization for flexible robot manipulation," in *IEEE-RAS International Conference on Humanoid Robots*, 2010.

[4] M. Stilman. and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *IEEE-RAS International Conference on Humanoid Robots*, 2004.

[5] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path planning among movable obstacles: a probabilistically complete approach," in *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[6] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *International Symposium on Experimental Robotics*, 2012.

[7] L. Chang, S. Srinivasa, and N. Pollard, "Planning pre-grasp manipulation for transport tasks," in *IEEE International Conference on Robotics and Automation*, 2010.

[8] J. King, M. Klingensmith, C. Dellin, M. Dogar, P. Velagapudi, N. Pollard, and S. Srinivasa, "Pregrasp manipulation as trajectory optimization," in *Robotics: Science and Systems*, 2013.

[9] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[10] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, "Physics-based grasp planning through clutter," in *Robotics: Science and Systems*, 2012.

[11] M. Gupta and G. Sukhatme, "Using manipulation primitives for brick sorting in clutter," in *IEEE International Conference on Robotics and Automation*, 2012.

[12] J. King, J. Haustein, S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *IEEE International Conference on Robotics and Automation*, 2015.

[13] M. Dogar and S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[14] J. van den Berg, P. Abbeel, and K. Goldberg, "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information," in *Robotics: Science and Systems*, 2010.

[15] N. Hyafil and F. Bacchus, "Conformant probabilistic planning via CSPs," in *International Conference on Automated Planning and Scheduling*, 2003.

[16] B. Bonet and H. Geffner, "Labeled RTDP: Improving the convergence of real-time dynamic programming," in *International Conference on Automated Planning and Scheduling*, 2003.

[17] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European Conference on Machine Learning*, 2006.

[18] J. Tsitsiklis, "On the convergence of optimistic policy iteration," *Journal of Machine Learning Research*, vol. 3, pp. 59–72, 2002.

[19] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Conference on Neural Information Processing Systems*, 2010.

[20] S. Thrun, "Monte Carlo POMDPs," in *Conference on Neural Information Processing Systems*, 2000.

[21] G.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A new framework for game AI," in *Artificial Intelligence Interactive Digital Entertainment Conference*, 2008.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[23] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. Weghe, "HERB: A Home Exploring Robotic Butler," *Autonomous Robots*, vol. 28, no. 1, pp. 5–20, 2010.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[25] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial Intelligence*, 1970.

[26] A. Johnson, J. King, and S. Srinivasa, "Convergent planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1044–1051, 2016.

[27] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000.

[28] A. Krontiris and K. Bekris, "Dealing with difficult instances of object rearrangement," in *Robotics: Science and Systems*, 2015.

[29] K. Seiler, H. Kurniawati, and S. Singh, "An online and approximate solver for POMDPs with continuous action space," in *IEEE International Conference on Robotics and Automation*, 2015.

[30] M. Koval, N. Pollard, and S. Srinivasa, "Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty," in *Robotics: Science and Systems*, 2014.