IEEE International Conference on Robotics and Automation (ICRA), 2013

Planning with Movable Obstacles in Continuous Environments with Uncertain Dynamics

Martin Levihn Jonathan Scholz Mike Stilman

***Abstract*— In this paper we present a decision theoretic planner for the problem of Navigation Among Movable Obstacles (NAMO) operating under conditions faced by real robotic systems. While planners for the NAMO domain exist, they typically assume a deterministic environment or rely on discretization of the configuration and action spaces, preventing their use in practice. In contrast, we propose a planner that operates in real-world conditions such as uncertainty about the parameters of workspace objects and continuous configuration and action (control) spaces.**

To achieve robust NAMO planning despite these conditions, we introduce a novel integration of Monte Carlo simulation with an abstract MDP construction. We present theoretical and empirical arguments for time complexity linear in the number of obstacles as well as a detailed implementation and examples from a dynamic simulation environment.

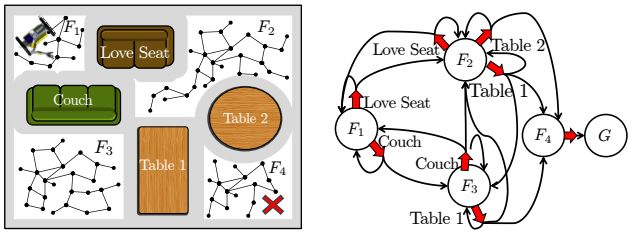
I. Introduction

One of the hallmarks of human motion planning is the ability to act based on *beliefs* about the dynamics of the objects in the environment. By contrast, state-of-the-art robot manipulation planners require exact, deterministic models of these dynamics in order to produce successful plans. This is a problem for many important tasks, such as Navigation Among Movable Obstacles (NAMO) [1–5], because it is unreasonable to expect robots to have perfect models of all possible objects in natural environments. As a result, planners for these tasks must be equipped for planning over a *dynamics belief space*, and for updating these beliefs with experience.

In [5] we presented a NAMO planner for discretized state and action spaces which was capable of taking uncertainty into account, and allowing the robot to bias its decisions towards plans likely to succeed. While theoretically promising, there are two fundamental challenges faced by real robotic systems that prevent the use of the planner. The first is the existence of continuous state and control spaces, which typically cannot simply be discretized without sacrificing either runtime or resolution. Second is the use of closed-loop controllers and state-space planners for manipulation on actual robots, as opposed to open-loop action primitives.

In this paper we address both challenges, and present a method for obtaining and solving the abstract MDP representation introduced in [5] for continuous state and action spaces using a realistic manipulation stack. We achieve this by combining modern sample-based planners with Monte Carlo simulation of object dynamics. As a result, our algorithm provides a decision-theoretic solution to action selection for the NAMO task on a continuously actuated robot.

The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: *levihn@gatech.edu, jkscholz@gatech.edu, mstilman@cc.gatech.edu*



(a) Robot determines free space regions as subgraphs in a PRM.

(b) Resulting MDP.

Fig. 1. Robot determines free space regions as subgraphs in a PRM and constructs MDP accordingly.

This work is organized as follows: following the outline of related work in Section II, Section III re-caps preliminaries. The general overview of the proposed method in Section IV is made concrete with details about our actual implementation in Section V. After a theoretical analysis and execution demonstrations in Section VI, the paper concludes with final remarks in Section VII.

II. Related Work

Navigation and manipulation planning poses a significant computational challenge even with *complete environment information*. Wilfong [6] first proved that deterministic NAMO with an un-constrained number of obstacles is NP-hard. Demaine et al. [7] further showed that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard.

In [1], Stilman et al. presented a planner that solved a subclass of NAMO problems termed LP_1 where disconnected components of free space could be connected independently by moving a single obstacle. The planner was able to solve the hard problems presented in [8] and was successfully implemented on the humanoid robot HRP-2 [4]. Our state space decomposition is mainly based on the concept of "free-space regions" introduced in [1]. However, the free-space concept in [1] was simply a rhetorical device, and has never actually been used in a NAMO algorithm. The work presented here takes direct advantage of the free-space representation. Subsequent work presented a probabilistically complete algorithm for NAMO domains [2]. However, all these methods solved NAMO assuming perfect knowledge of the environment and deterministic action outcomes.

Wu [3] et al. and Kakiuchi et al. [9] introduced the first extensions to NAMO in *Unknown Environments*. In [3] a planner was presented that could solve NAMO problems given

only partial obstacle information. However, that planner is not capable of handling uncertainty, and instead assumes that everything within sensor range is ground truth. Further, actions are again assumed to be deterministic. [9] presented a system that executes NAMO in unknown environments on the humanoid robot HRP-2. However, the authors took a reactive behavior-based approach, rather than attempting full decision theoretic planning.

In [10] Dogar et al. presented a planning framework capable of rearranging clutter admits uncertainty. However, the work is mainly focused on object pose uncertainty rather than significant uncertainty in the manipulation dynamics. In addition, while the framework explicitly represents the existing uncertainty and computes plans that are robust to it, the framework does not reason about which objects to manipulate in order to increase the success probability. In contrast, the presented work allows the robot to bias its decisions at plan time in order to compute policies that are likely to succeed.

III. Preliminaries

The presented algorithm is based on the theory of an MDP and Monte Carlo simulation. This section briefly introduces the appropriate definitions.

A. Markov Decision Processes

The Markov Decision Process (MDP) is a model for stochastic planning, and the foundation of our algorithm. We define an MDP $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$ for a finite set of states S, a finite set of actions A, a transition model $T^a_{ss'} = P(s'|s, a)$ specifying the probability of reaching state s' by taking action a in state s, a reward model $R^a_s = r(s, a)$ specifying the immediate reward received when taking action a in state s, and the discount factor $0 \leq \gamma < 1$.

The standard technique for solving MDPs is *Value Iteration* [11], which is used to find the optimal policy $\pi^* : f(s) \rightarrow a$ which maps states to actions in order to maximize the expected long-term reward, or value $V(s)$ for all states $s \in S$. In general, value iteration (VI) has a polynomial runtime in the number of states (with fixed error ϵ).

B. Monte Carlo Simulation

Monte Carlo simulation is a statistical tool relying on the law of large numbers to numerically solve problems that are difficult or impossible to solve analytically. To obtain an output measure for a function $f = H(\mathbf{X})$ with dependent parameters $\mathbf{X}$ governed by a probability distribution $P(X)$, Monte Carlo simulation precedes by repeatedly sampling a specific vector $X \sim P(X)$ and obtaining the output value $f = H(X)$. Statistical inference is then performed on the obtained output values [12].

In the following sections we demonstrate how these two very different techniques can be combined to obtain a NAMO planner for continuously actuated robots admits uncertainty in the manipulation dynamics.

IV. Approach

The next section provides a general overview of the proposed planner. The goal of our algorithm is to achieve decision-theoretic planning in the NAMO task for *dynamically situated* robots. Here the term "dynamically situated" refers to both the existence of continuous state and control spaces, as well as the customary stack of closed-loop controllers and planners which have been developed to accommodate these situations.

We approach this problem by building on previous work, which introduced an "approximate" MDP that closely resembles the real problem but can be solved in linear time for typical environments [5]. The construction of this MDP builds on two insights of the domain. First, there is a natural abstraction from the low-level state space into a small set of abstract states, which we call "free-space regions" to indicate that the robot can move collision-free between any two configurations within the region. This suggests also a small number of implied abstract actions for maneuvering in this abstract state space: since each free space region is circumscribed by obstacles, we can define the abstract action "create an opening to neighboring free-space" for each obstacle. This property is the basis for the state and action spaces in the NAMO MDP, visualized in Figure 1.

The second insight is that we can capture the transition and reward dynamics in this abstract representation by computing a low-level manipulation policy for each abstract action. The transition model, $T^a_{ss'}$, is used to model uncertainty in *manipulation dynamics*, such as might occur if obstacle mass or friction is unknown, and should reflect the likelihood of actually creating an opening between the free spaces. The reward function, R^a_s, reflects the expected reward (or cost) in terms of required time and physical work for creating such an opening.

Together these two ideas permit the construction of a hierarchical MDP, which can be solved in a manner analogous to MAX-Q [13]. The obtained policy describes both a mapping for the abstract representation – from a given free space region to an object to manipulate, as well as the raw representation – from a given obstacle state to a control vector.

V. Implementation

To construct the proposed NAMO planner we must provide appropriate methods for deriving all components of the MDP $(S, A, T^a_{ss'}, R^a_s)$. However, none of the key ideas from [5] transfer directly to the continuous case for dynamically situated robots. The method for free-space detection relies on a discrete state space, and the use of Monte-Carlo Tree Search (MCTS) for manipulation planning requires both a discrete state space as well as a finite set of open loop transition primitives. In this section we address each of these challenge in turn.

A. States and Actions

The efficiency of our MDP approach depends on efficient methods for identifying the independent free space regions

Algorithm 1: NAVIGATE_NAMO

Input: W: world, k: number samples for Monte Carlo simulation, c_g: goal, $oldMDP = \emptyset$: previous MDP if any

```
// determine MDP:
1  MDP ← GET_STATES_AND_ACTIONS(W);
2  COPY_VALUES(oldMDP, MDP);
   // perform value iteration:
3  while something changes do
4      for s ∈ S do
5          if getFs(s) contains goal then
6              s.v ←goal reward;
7              continue;
8          s.v ← 0;
9          for a ∈ getActions(s) do
10             att ← [];
11             for i ← 0 to k do
                   // sample physical param:
12                 a.obj.pp ← SAMPLE(P(Ω));
13                 att[i] ←CONNECT_FS(a.obj, a.toFs)
14             s' ← getState(a.toFs);
15             succ ← {succesfull plans ∈ att};
16             T^a_{ss'} ← |succ|/k;
17             R^a_s ← 1/|succ| Σ_{s∈succ} αF(s)+βτ(s)+γT(s)
               q_a ← T^a_{ss'} R^a_s + γs'.v;
18             if q_a > s.v then
19                 s.v ← q_a;
20                 π(s) ← a;
   // fill robot task queue:
21 s_r ← getState(robot.config);
22 robot.addTask(navigateTo(π(s_r).graspPos));
23 robot.addTask(executeManip(π(s_r)));
24 robot.addTask(updateDistributions());
25 robot.addTask(NAVIGATE_NAMO(W, k, c_g, MDP))
```

and their associated actions. In [5] the state-space was discretized, which allowed an $O(logN)$ Dijkstra's approach ("wavefront expansion"), where N represents the number of states. For a typical robot in continuous state and control spaces, we have two options. First, we could simply discretize and re-use the wavefront approach. However, as shown in [14], producing reasonable behavior on real robots would require far too many states. Instead, we adopt a sampling approach that has been proven in the planning literature and build a roadmap over the state space. The main insight behind this approach is that the resulting roadmap will contain disconnected subgraphs for precisely the free space regions that we are attempting to identify.

PRM State Clustering: The probabilistic roadmap (PRM) [15] algorithm samples random configurations within the state-space and connects nearby collision-free samples if there is a collision-free path between them. Constructing a PRM in a disconnected configuration space will consequently return multiple disconnected subgraphs. Each of these subgraphs now encode separate free space regions and together they yield the MDP states. This is visualized in Figure 1.

Having determined the MDP states, the actions need to be determined. This is done by finding the obstacles disconnecting two free space regions. We accomplish this with a slight extension to the PRM construction phase: Instead of only considering random state-space samples during the construction of the PRM, we also use samples of valid grasping poses for objects. If, upon termination of the PRM construction phase, sampled grasping poses of the same obstacle belong to different subgraphs, the obstacle is considered disconnecting the free spaces and an according action in the MDP is created. This is summarized in the function call in line 1, Algorithm 1.

B. Transitions and Rewards

In [5] we showed that for discrete state-action spaces we can compute ϵ-optimal manipulation policies using MCTS, which is limited by the planning horizon rather than the total number of states. By construction, these policies provided estimates of the transition probabilities and rewards for each action in the free-space MDP. Here we introduce an alternative to MCTS which is appropriate for the case of dynamically situated robots.

The Object Property Model: The main idea is to represent *transition* uncertainty close to its source as *object property* uncertainty, which we capture using distributions over the relevant physical quantities. We define Ω as the minimal set of quantities governing object dynamics on a plane, and include mass and one or more anisotropic (wheel) constraints. Importantly, Ω should be viewed as a set of auxiliary variables for predicting transition dynamics, and not a state parameter itself.

To understand this as a stochastic transition model, consider the special case of zero-uncertainty where $P(\Omega) = \delta_\Omega$, with δ_Ω denoting the dirac delta function on a particular assignment to Ω. Here the model reduces to a deterministic environment governed by rigid body physics and the chosen controllers. Consequently, the transition model reduces to 0 or 1 and the reward function becomes a deterministic function of the physical work required by the robot to manipulate an object to create an opening. For the general case $P(\Omega)$ is an arbitrary distribution over Ω (e.g. multivariate-normal), we may generate samples of the transition dynamics in the NAMO MDP by sampling object parameters from $P(\Omega)$. Repeating this process and performing statistical inference over the obtained transition and reward models represents a stochastic simulation with outputs that reflect the expected interaction between the possible physical properties and the robot.

Our particular choice of parameters aims to satisfy the LP_1 class of NAMO problems [1], where regions are separated by individual, disconnected obstacles. We therefore require parameters to describe rigid-body dynamics for non-coupled objects, such as carts, chairs, tables with lockable wheels and casters. The central aspect governing the dynamics of this class of objects is contact with the ground. This

includes isotropic friction forces, such as table feet, and more generally anisotropic friction forces to accommodate wheels. Other types of constraints, such as prismatic and revolute joints, serve to couple multiple bodies, and remain a challenge for future work.

In 2D, an anisotropic friction joint between the ground and a point on the target object can be represented with five new parameters per joint: $\{x, y, \theta, mu_x, mu_y\}$, corresponding to 2D pose and orthogonal friction coefficients. For typical wheels, one of these friction coefficients is close to zero, and the other close to one. Furthermore, two joints are sufficient to describe any wheeled body in 2D, because any two joints at a given orientation can be expressed as a single joint along their common axis, and more than two joints at unique orientations serve to fully constrain the system (we do not consider what happens when constraints are violated). This means that the transition dynamics of arbitrary disconnected obstacles in 2D, including shopping carts, wheelchairs, and tables with lockable wheels, can be fully specified with $1 + 5 \times 2 = 11$ parameters. The first parameter is reserved for mass, which affects friction forces as well as the overall effort of manipulation.

Of these, mass m can take values in $\mathbb{R}^+$, friction coefficients μ_x, μ_y can take values in $[0, 1]$, position parameters x, y can take values within the bounds of the object $[a, b]$, and orientation can take values in $[-pi, pi]$. To represent the robot's beliefs over these parameters, we assign the following distributions:

- $m \sim log\text{-}normal(\mu_m, \sigma_m)$
- $\mu_x, \mu_y \sim truncated\text{-}normal(\mu_f, \sigma_f^2, 0, 1)$
- $x, y \sim truncated\text{-}normal(\mu_p, \sigma_p^2, a, b)$
- $\theta \sim von\text{-}mises(\mu_t, \kappa_t)$

Using the object property model, it is possible to solve the NAMO MDP for dynamically situated robots. Recall that the transition function T_{hl} for the NAMO MDP encodes the probability of successfully creating an opening and the reward function R_{hl} the expected reward (or cost) in terms of required time and physical work for creating such an opening. While our previous work for discretized environments in [5] obtained estimates for these quantities by introducing and sampling a low-level MDP, our solution method for continuous environments builds on Monte Carlo simulation to obtain estimates of T_{hl} and R_{hl} directly. The trade-off is that direct simulation discards all intermediate results, and maintains no policy information for visited states.

To see how estimates of T_{hl} and R_{hl} can be obtained from Monte-Carlo simulation, let $T^a_{ss'}$ and R^a_s represent the specific instance of the transition and reward function for action a and states s and s'. To obtain value estimates, we perform the following evaluation k times[1]:

1) Sample world w with object parameters $\omega \sim P(\Omega)$ for all objects.
2) Call a manipulation planner on w trying to create an opening between the free-spaces represented by s and s' using the object represented by a and save the result.

[1] k may be a fixed value or a function of the degree of uncertainty

$T^a_{ss'}$ is now set to be the ratio of manipulation plans that succeeded in creating an opening.

$$T^a_{ss'} = P(s' = target|s, a) = \frac{|succ|}{k} \tag{1}$$

R^a_s is set to:

$$R^a_s = \frac{1}{|succ|} \sum_{s \in succ} \alpha F(s) + \beta \tau(s) + \gamma T(s) \tag{2}$$

where $succ$ denotes the set of successful manipulation plans, $F(\cdot), \tau(\cdot), T(\cdot)$ functions returning the force, torque and time required by a specific plan respectably, and α, β, γ representing weights. (Note that we can not just fix a manipulation plan and evaluate it for different samples of $P(\Omega)$ as this would yield an estimate of the success probability of a specific plan rather than provide insight into the general success probability of manipulating the object.) Given these estimates, M_{hl} can be solved using standard value iteration. Algorithm 1 summaries the solution approach and the following section demonstrates a concrete example implementation.

C. Implementation Details

While the discussion so far has presented a general approach, we now present our specific implementation of the Monte Carlo simulation for obtaining $T^a_{ss'}$ and R^a_s. The reader is encouraged to keep in mind that this is an example implementation and is likely to be adapted to the specific robot system and environment at hand.

Model Sampling: In our implementation we assume independence between the individual object parameters and obtain samples of the joint distribution $P(\Omega)$ by sampling each parameter independently according to its distribution.

Manipulation Planning: Given a sample of $P(\Omega)$, it has to be determined if, and at what cost, an opening between the free spaces represented by s and s' can be created. To allow planning with arbitrary constraints on the obstacles and robot, we implemented a kinodynamic-RRT (RRT_{KD}) planner [16]. RRT_{KD} operates in the phase space of the robot – configuration and velocity components for each degree of freedom – and searches by sampling its control space. The RRT_{KD} search terminates if either a number of maximum nodes m have been reached or an opening has been created[2].

Unfortunately, the verification of an opening is a non-trivial task in itself, and is therefore only performed every t node expansions. We perform opening verification by checking the existence of a path between the free spaces using a low-dimensional RRT [17] (RRT_{FP}) that only considers the robot's footprint and is limited to a maximum number of nodes (alternatively one could use a visibility graph planner [18] to avoid false negatives caused by the node limit). The start configuration for RRT_{FP} is given by the robot's

[2] m should be chosen to be small as the creation of an opening is typically a very local manipulation. It can also be set based on the value of the goal free space to take into account the value of creating such an opening, similar to the dynamic horizon in [5].

configuration in the node in RRT_{KD} that triggered the opening verification while the goal configuration is set to be a random configuration within the goal free space. If a path is found, an opening is reported. The astute reader will observe that openings might also be gleaned from the roadmap graphs. However, this is only possible if the graphs are maintained in synchrony with the manipulation planner, which we determined to be too expensive to do in practice.

In Algorithm 1, these steps of the Monte Carlo simulation method are summarised in lines 10-11.

Computing Transition and Rewards: $T^a_{ss'}$ and R^a_s are now set according to Eq. 1 and Eq. 2, respectively.

D. Execution

Recall that value iteration requires state and action sets S and A, as well as transition and reward functions $T^a_{ss'}$ and R^a_s. Section V-A describes how to obtain S and A by analyzing the output of a PRM. Eq. 1 describes the generative transition dynamics at the free-space level, and Eq. 2 the reward for free-space actions. With these quantities defined it is possible to perform value iteration on the free-space MDP.

This yields a policy over the free space regions. The robot then directly executes the action defined for its current free space: it navigates to the grasping position and executes the manipulation. After the execution of the manipulation plan, the robot may update its distributions based on the interaction with the object. In our implementation, we prototyped this "learning" step by updating the probability of rotational constraints for rotation-only events: if after an applied off-axis force the body merely rotated about its center, we increase the probability of a rotational constraint about its center of mass. Future work will investigate the use of more general statistical inference techniques over the entire object property representation.

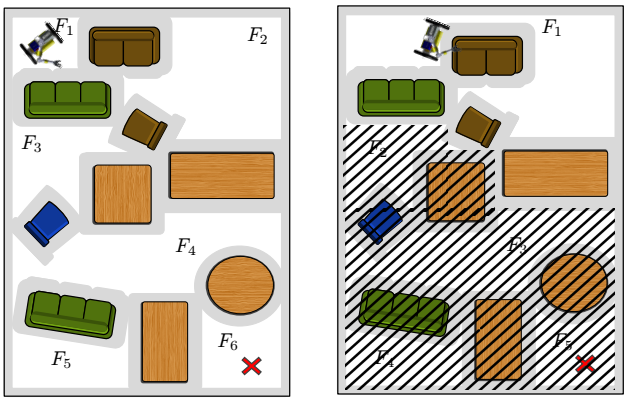
The robot then recursively calls Algorithm 1, lines 21-25. To avoid intensive re-computations in the recursive call, our algorithm attempts to reuse information by copying simulation results for all actions leading to and from unchanged states. To identity these states, we propagate a "clean" bit outwards from the goal state until a state is reached that does not have a corresponding state in the new environment configuration. Figure 2 shows an example of duplicated information. Notice that information associated with objects adjacent to the changed free space is not preserved as the manipulation plans might not be feasible anymore. This is done in the function COPY_INFO 2, Algorithm 1.

The following section argues for the usefulness of the proposed framework.

VI. Evaluation

We have implemented a prototype of the proposed framework in a simulation using Python and the open source 2D physics engine pyBox2D [19]. The robot is modeled as a nonholonomic 5 Degree of freedom (DOF) mobile manipulator with a 3 DOF base and a 2 DOF arm.

Prior to demonstrating execution examples at the end of the section, we provide a general complexity analysis of the proposed framework and present obtained runtimes. Recall that the difficulty of the NAMO domain is caused by the exponential dependence on the number of movable obstacles. We show that for non-degenerate environments the proposed framework is linear in the number of movable obstacles.



(a) Configuration prior to execution. (b) After execution. Information within shaded area is duplicated.

Fig. 2. Robot determines information to copy to new MDP.

A. Complexity

The proposed framework executes value iteration on the free space MDP. Within each loop of value iteration we perform Monte Carlo simulation over a sample-based planner (RRT). The run-time of the Monte Carlo simulation depends on the number of iterations k, a constant, and the chosen manipulation planner. In general, the run-time of the manipulation planner depends on the size of the state space of the robot and the obstacle to manipulate. However, as all other obstacles are considered to be static during manipulation planning, it is not a function of the number of movable obstacles in the environment (also recall that we have a node limit). The complexity of the Monte Carlo simulation is consequently constant in the number of movable obstacles $|O|$ and the complexity of the framework is governed by the execution of value iteration on the free space MDP.

The size of the state space of the free space MDP is linear in $|O|$ (worst case $2|O|$ for infinite obstacles intersecting in a star pattern). While the complexity of value iteration is polynomial in the size of the state space in general, it is linear if the transition matrix T has *on average* a constant number of next-states with non-zero probability. In [5] we showed that T for the free space MDP fulfills this requirement if the *adjacency graph* G over free space regions is planar. G has vertices for every free space region and edges between vertices if the free spaces are disconnect through a single obstacle. This is generally fulfilled for non-degenerate environments.

In conclusion, the proposed framework has a time complexity *linear* in the number of obstacles.

In addition to the theoretical complexity analysis, we performed a preliminary empirical evaluation of 70 trials with varying environment sizes, objects, object placements, and object parameter distributions. The goal configuration

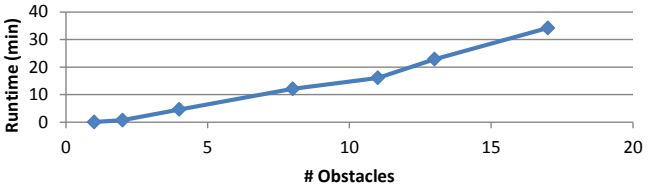


Fig. 3. Average runtimes for 70 trails.

was always in a different free space region than the initial robot configuration. Figure 3 summarizes the runtime as a function of the number of obstacles. These preliminary experiments with manually designed environments confirm linear dependence. The obtained runtimes are in the range of minutes, however orders of magnitudes of speedup can be achieved for more efficient programming languages, collision detection algorithms and parallelisation of the Monte Carlo simulation runs. Given the linear relationship, such constant factors present the bottleneck rather than the algorithm itself and are the subject of our future work. These results provide a general argument for the usefulness of the proposed framework. We now detail some execution examples.

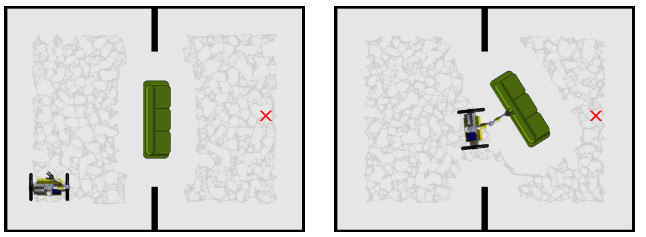
B. Results

While the accompanying video demonstrates different execution trials with up to 30 obstacles, this section highlights some example behaviors of our planner.

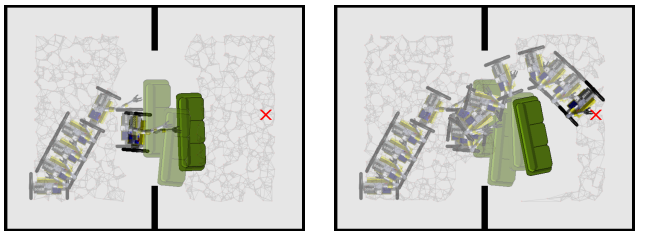
1) Uncertain Manipulation: Figure 4(a) shows the initial configuration of a simple environment. The robot has to reach the goal, but the couch prevents it from reaching the free space containing the goal. In addition, there is uncertainty associated with the parameters of the couch. The robot executes our proposed framework, beginning by constructing a PRM over the space, shown in light gray. After determining the couch to be the only obstacle whose successful manipulation would clear the goal, the robot performs Monte Carlo simulation over manipulation plans for the couch. It then chooses a manipulation plan with the expected final couch configuration shown in Figure 4(b). If the couch would indeed result in the expected configuration after the manipulation, the robot could circumvent the couch on the bottom and reach the goal.

However, because the exact object parameters are unknown to the robot, the execution of the plan instead results in the configuration shown in Figure 4(c). Realizing that the required connectivity of the free spaces has not been reached, the robot re-computes based on the new configuration. The robot decides instead to push the couch down a bit and move past it above. Figure 4(d) shows that the robot now successfully clears the goal.

2) Decision Update: Figure 5(a) demonstrates a similar setup but with multiple disconnecting obstacles. The robot again has uncertainty about the objects, but this time also begins with low probability that any of the objects are constrained. It decides to move the lighter table. The expected outcome of the chosen manipulation plan is shown in Figure 5(b). However, as the table is in fact constrained to only rotate, the robot instead finds itself in the configuration

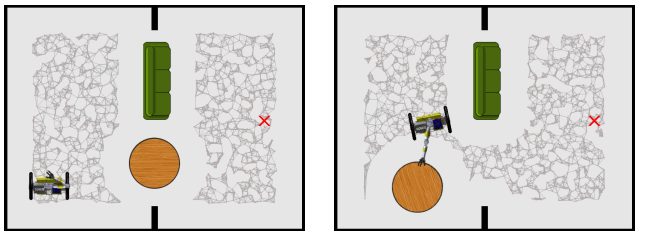


(a) Initial configuration (b) Expected outcome: robot could drive below the couch.

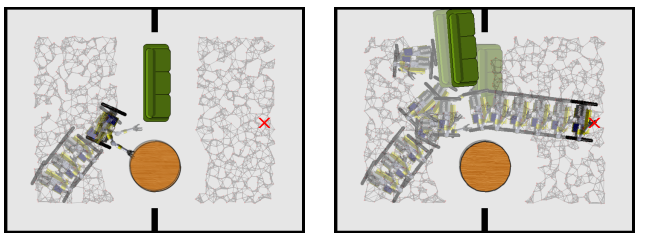


(c) Actual behavior: robot recomputes based on new configuration. (d) Moves couch further down and circumvents above.

Fig. 4. Robot clears goal despite couch behaving unexpectedly.



(a) Initial configuration (b) Expected outcome: table has low probability of being constrained.



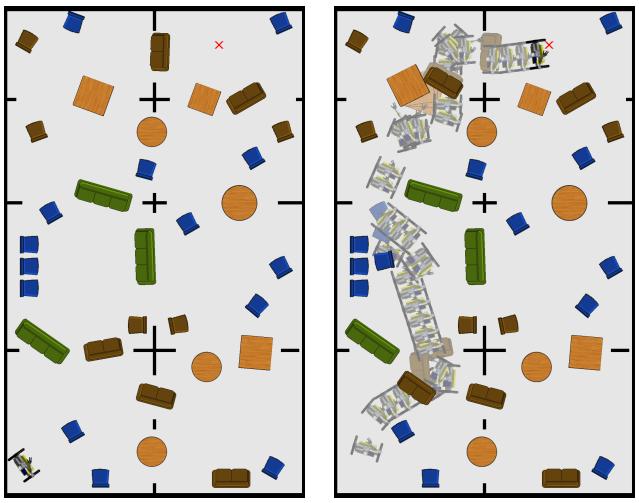
(c) Actual behavior: table is constrained. Robot increases probability of table being rotationally constrained. (d) Based on the new information the robot decides to move the couch.

Fig. 5. Robot incorporates new information to enhance the plan.

visualized in Figure 5(c). The robot detects that, despite the applied force, the object has only rotated, not moved. The robot updates its belief about the table being rotationally constrained to 98%. Given this new information, the subsequent plan involves moving the couch instead. This finally allows it to successfully clear the goal as visualized in Figure 5(d).

While general inference techniques for updating the distributions given observed object behaviors is future work, these examples show that our framework allows the robot to use updated belief distributions.

3) Large Execution Example: Figure 6 shows an execution example with more than 30 obstacles. The accompanying video demonstrates further execution examples.



(a) Initial configuration. (b) Execution path of the robot.

Fig. 6. Execution example with more than 30 obstacles.

VII. Conclusion

In this paper, we have presented a decision-theoretic NAMO planner that is applicable to modern robotic systems. The planner is capable of taking uncertainty about object parameters into account while guaranteeing run-time linear in the number of objects. To achieve this we integrated an established NAMO MDP formulation with Monte Carlo simulation of world dynamics. The use of a full physics engine as the core of a stochastic dynamics model is key to our ability to model arbitrary obstacle interactions. We feel that our combination of stochastic planning and physical simulation is an important step towards the flexibility necessary to operate in natural environments.

We demonstrated a successful implementation operating in continuous state and action spaces for a non-holonomic robot and obstacles and have shown unprecedented behavior in the NAMO domain.

In future work, we will develop more general inference methods for updating the robot's beliefs over object parameters by observing their actual reactions to applied forces. In addition, we will incorporate pose uncertainty, and develop full belief-space planners which are grounded in rich priors on the physical behavior of objects.

Acknowledgments

This research was supported by NSF grant IIS-1017076.

References

[1] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Journal of Humanoid Robotics*, 2004, pp. 322–341.

[2] J. Van Den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path planning among movable obstacles: a probabilistically complete approach," *Workshop on Algorithmic Foundation of Robotics*, 2008.

[3] H.Wu, M. Levihn, and M. Stilman, "Navigation among movable obstacles in unknown environments," in *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 10)*, October 2010.

[4] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, "Planning and executing navigation among movable obstacles," in *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, October 2006.

[5] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical decision theoretic planning for navigation among movable obstacles," in *Workshop on the Algorithmic Foundations of Robotics (WAFR'12)*, June 2012.

[6] G. Wilfong, "Motion planning in the presence of movable obstacles," in *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*. New York, NY, USA: ACM, 1988, pp. 279–288.

[7] E. Demaine, J. O'Rourke, and M. L. Demaine, "Pushpush and push-1 are np-hard in 2d," in *In Proceedings of the 12th Canadian Conference on Computational Geometry*, 2000, pp. 211–219.

[8] P. Chen and Y. Hwang, "Practical path planning among movable obstacles," in *In Proceedings of the IEEE International Conference on Robotics and Automation*, 1991, pp. 444–449.

[9] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, "Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[10] M. Dogar and S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, pp. 1–20, 2012.

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[12] C. Lemieux, *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer Verlag, 2009.

[13] T. Dietterich, "An overview of maxq hierarchical reinforcement learning," *Abstraction, Reformulation, and Approximation*, pp. 26–44, 2000.

[14] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[15] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[16] S. LaValle and J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[17] J. Kuffner Jr and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[18] T. Lozano-Pérez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[19] "pybox2d," http://code.google.com/p/pybox2d/, June 2012.