2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Navigation Among Movable Obstacles with Learned Dynamic Constraints

Jonathan Scholz[1] Nehchal Jindal[2] Martin Levihn[2] Charles L. Isbell[2] Henrik I. Christensen[2]

***Abstract*— In this paper we present the first planner for the problem of Navigation Among Movable Obstacles (NAMO) on a real robot that can handle environments with under-specified object dynamics. This result makes use of recent progress from two threads of the Reinforcement Learning literature. The first is a hierarchical Markov-Decision Process formulation of the NAMO problem designed to handle dynamics uncertainty. The second is a physics-based Reinforcement Learning framework which offers a way to ground this uncertainty in a compact model space that can be efficiently updated from data received by the robot online. Our results demonstrate the ability of a robot to adapt to unexpected object behavior in a real office scenario.**

I. Introduction

Creating robots that can efficiently maneuver in domestic environments, such as homes and offices, is an important long-term goal in robotics. If the robot is capable of manipulating obstacles to clear paths to the goal, this problem is referred to as Navigation Among Movable Obstacles (NAMO). However, unlike factories and laboratories, it is unreasonable to expect that these environments can be accurately specified in advance. Therefore a key challenge is to allow the robot to cope with the inevitable inconsistencies between its expectations and the actual behavior of the environment when a plan is executed. This paper presents a Reinforcement Learning approach to this problem, in which the robot is formalized as an agent that acts based on beliefs about the dynamics of objects in the environment, and adapts when its expectations are violated.

Our work draws on two threads from the RL literature. The first is the NAMO-MDP [7], [8], a hierarchical Markov-Decision Process framework for solving NAMO tasks with stochastic dynamics models. The second is Physics-Based Reinforcement Learning (PBRL) [14], which provides a concrete parameterization of dynamics uncertainty in terms of probability distributions over physical parameters, such as mass and friction, inside a physics engine describing the scene. Although promising, the usefulness and scalability of these methods on an actual (non-simulated) NAMO task have not yet been demonstrated.

In order to integrate these methods into a practical planner on a real robot, the key challenge is to provide a solution for the low-level manipulation tasks that does not break the hierarchical abstraction of the NAMO-MDP, but can handle the large and highly constrained control space of the robot-object system. We introduce two methods for addressing this challenge. The first is a whole-body manipulation controller which abstracts away the robot's body and presents a low-dimensional action space in terms of the manipulated object. The second is a heuristic for searching this action space that is encoded as a *model-dependent manipulation policy*. This policy representation is conditional on the world state *and* the parameters of the target object, and serves to guide search towards actions that are appropriate for the dynamics of the target object.

This paper is organized as follows. Section II provides an overview of related work. Section III summarizes the two building blocks of this work, *Physics-Based Reinforcement Learning* and Markov Decision Processes for *Navigation Among Movable Obstacles*. Section IV and Section V-C present our method for whole-body manipulation with learned object models, and we evaluate our method in Section VI. We provide closing remarks and discuss future work in Section VII.

II. Related Work

Approaches to whole body manipulation can be distinguished based on two key properties: whether they consider kinematic or dynamic constraints on the *robot* , and whether they consider kinematic or dynamic constraints on the *target object* . In this work we consider a kinematically-constrained robot interacting with objects that have unknown dynamic constraints. Dynamically-constrained robots, such as bipeds, must take into account the robot's reaction to applied forces in a balancing controller. These platforms offer the possibility of exploiting the robot's dynamics to increase the amount of applied force [15], but in practice this comes at the expense of stability and safety. Kinematically-constrained objects are ubiquitous in human environments, for example doors, drawers, utility carts, and tables with lockable casters. Planning with kinematically-constrained objects has recently become a popular topic in robot manipulation.

[12] provides an example of planning for an unconstrained object (a cart) with an unconstrained robot base (the PR2). Even in this simple case, the robot+cart system is kinematically constrained via the grasp point, and the authors resort to graph-search over articulation primitives to achieve long-horizon mobile manipulation.

An example of a dynamically-constrained robot interacting with kinematically constrained objects can be found in [2]. This is similar to the problem we consider here, but was

1. Google DeepMind, London, UK. Email: *jscholz@google.com*
2. Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: *nehchal@gatech.edu, isbell@cc.gatech.edu*

implemented on a *NAO*, which is a small bi-pedal robot that is dynamically stable but incapable of reaching and pushing large objects such as office furniture. The planner in [2] focused on maintaining balance while tracking a constrained gripper trajectory with a single arm, and only utilized torso and leg DOF to maintain static balance. The authors considered two constraint classes, prismatic and revolute, and assumed that both the constraint and the corresponding gripper trajectory (*e.g.* a circular arc) are known a priori. By contrast, we consider bi-manual manipulation while navigating with a non-holonomic base. Further, instead of strict kinematic constraints we consider anisotropic friction, a second-order constraint on object pose that acts via forces, and can therefore model slippage from idealized kinematic trajectories.[1] Thus instead of assuming predefined parametric trajectories, we obtain object trajectories by integrating object pose under applied manipulation forces using a physics simulator that is parameterized by the robot's current beliefs.

Another important distinction is that the results in [12] and [2] are concerned with only a single manipulation action. The goal of this work is to utilize these manipulation capabilities to solve long-horizon tasks with multiple grasp and manipulation actions. This motivates the contributions in Section V-C.

Another major theme of this paper is online model learning. Learning the parameters of dynamical systems is a well-studied topic in machine learning and control theory, and there are numerous approaches including non-parametric regression [3], spectral learning for subspace [16] or predictive-state [1] representations, Bayesian model selection [10], and many more. Here we concern ourselves with physics-based scene-level dynamics where the primary focus is constrained object movement. We summarize the relevant work in this area, and for a broader review see [14].

Like the physics-engine-based method considered here, [20], also considers fitting object masses in a physics engine using video data. However, this model lacked any notion of physical agent, which implicitly assumes that the only applied force is gravity. By contrast, our approach considers control input from a robot, in the form of applied forces and torques, as a critical component of the inference process. This is similar to [11] and [17], which learn action-conditional latent dynamics representations from pixel inputs. However unlike [11], [17], we focus on explicit physics-based model spaces for two reasons. The first is sample efficiency – both pixel-based methods had to be trained on millions of frames to reach adequate performance. Second, as we will see in Section IV-C, the parameters in a physics-based model space carry specific semantic meaning that can be utilized for efficient planning and control.

III. Preliminaries

This work builds chiefly on two methods from the machine learning and robotics literatures, which we summarize here.

[1]This "softness" also turned out to be useful for estimation because it smooths the gradient of loss function on object parameters.

A. Physics-Based Reinforcement Learning

Physics-based Reinforcement Learning (PBRL) is a model-based RL framework specifically designed for agents interacting with physical objects [14]. The general idea in PBRL is to capture the robot's uncertainty about world dynamics as probability distributions over the dynamics parameters of a physics engine describing the scene. The main advantage of this approach is that the model representation is concise, allowing the agent to quickly estimate the model and generalize accurately.

To update a PBRL model, the robot gathers data by applying in-plane manipulation forces to one or more points on the object, and records the force-torque responses and resulting object trajectory. To avoid having to introduce the end-effector contact point(s) in the model, the sensor readings can be adjusted to compensate for the weight of the end-effector, and transformed to the object frame in real-time based on the current gripper pose. These operations are discussed in greater detail in [13].

The world-frame object trajectories and applied forces provide all the inputs necessary to simulate a full manipulation episode. Defining the model loss as the L2-norm on the integrated trajectory error, as discussed in [13], the model parameters can be estimated with standard non-convex inference techniques, *e.g.* MCMC [9] or L-BFGS [21].

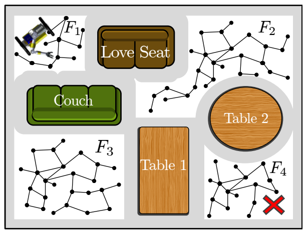
B. Navigation Among Movable Obstacles

The Navigation Among Movable Obstacles (NAMO) problem is like standard navigation problems, with the caveat that the path may be obstructed by *movable* obstacles that the robot can choose to clear. Unlike other mobile manipulation tasks such as cart pushing [12], the manipulation tasks in NAMO are underspecified: given a map of the room, how does the robot decide which path to take, or which object to move? If the final positions of all objects are unspecified, this leads to a problem dimensionality that is exponential in the number of objects in the scene, and is known to be NP-hard [18].

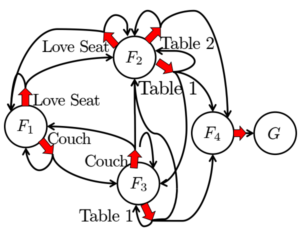
In addition to the dimensionality problem, practical solutions must handle uncertainty about the dynamics of the manipulable objects. [7], [8] presented the first stochastic planning method for solving NAMO problems in realistic environments with model uncertainty. The core technical idea behind this method is the NAMO-MDP, a hierarchical MDP model for capturing the abstract subproblem of moving between *free space regions*.

The NAMO-MDP defines a two-level hierarchy, with moving between regions as the abstract task, and clearing individual obstacles as the low level task. An action in the abstract MDP involves manipulating an obstacle to clear a path to a neighboring free-space region. This property is the basis for the state and action spaces in the NAMO MDP, visualized in Fig. 1.

The high-level NAMO-MDP can thus be viewed as a reward shaping mechanism to focus the low-level search on actions that are likely to clear paths to useful locations. This problem decomposition attains its savings by constraining



(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.

actions to affect only one object at a time, and collapsing all states within a free-space region to a single value.

The Q-value of a NAMO action represents the sum of two quantities: the cost of executing the manipulation action, and the reward of the target free-space region if the action was successful. In order to compute this high-level Q-value, a planner must be called to search for a manipulation action that can clear the target obstacle. This low-level manipulation problem defines another MDP, whose dynamics and action space depend on the robot and environment.

In earlier simulation-based work the low-level manipulation problems could be solved with Monte-Carlo Tree Search (MCTS) methods. Specifically, the discretized grid case was solved using vanilla MCTS over discrete push actions [5], and the continuous dynamically-constrained case was solved using Monte-Carlo simulation of KinoDynamic-Rapidly Exploring Random Trees (KDRRT) [6]. We found that these methods were not viable for directly obtaining whole-body controls on a high-dimensional physical robot. MCTS requires discretizing the configuration space, and is infeasible on sparse-reward tasks such as NAMO without some exogenous search bias. In our preliminary experiments we found that even KDRRT was not very efficient, and often produced low-quality plans when the model was not known with high accuracy [8]. This motivated a more constraint-centric approach, coupled with a policy-based search bias, which we present next.

IV. Approach

The raw control space of humanoid robots can be quite large. For example, the robot utilized below has a differential-drive base and two arms, yielding a total of $3+7+7=17$ DOF.[2] However, while actually manipulating an object, the dynamics of the robot-object system are highly constrained by the rigid coupling of the arms and the possible constraints on the motion of the object itself (*e.g.* a cart or a table with a locked wheel).

In this paper, we are primarily concerned with manipulation tasks involving pushing or pulling large objects on flat surfaces. In order to achieve coherent object motion, we must therefore search over control inputs that respect the following constraints:

1) Both grippers remain in-plane.
2) Both grippers produce the same twist in the object frame.
3) Overall gripper and base motion respects object constraints.

In addition to these hard constraints, it is also desirable that the grippers remain in their reachable workspace, and Section IV-A discusses a method for satisfying this constraint by shifting work from the arms to the robot base.

Our approach is to first reduce the dimensionality of the action space by defining a whole-body manipulation controller that abstracts away the robot DOF and provides an action space in terms of the manipulated object. This controller induces a planning problem in terms of the object grasp point and the desired object velocity. We then define a heuristic for planning in the form of a stochastic manipulation *policy*, which biases action selection during planning according to actions that are likely to be valid for the target object.

A. Constrained Object Manipulation

Like many humanoid robot systems, we assume access to low-level current and velocity controllers for the individual wheels and joint motors, but not necessarily torque control.[3] Our goal is therefore to achieve velocity control of *grasped objects* in terms of velocities at the wheel motors and the joints in the body and arms.

The starting point for our approach is to define a whole-body Jacobian pseudo-inverse controller which computes base and gripper velocities in the robot-frame from desired object velocities in the object-frame.[4]

A body Jacobian for the planar robot-object system is an 3×9 matrix mapping robot and gripper velocities in the robot frame to object velocities in the object frame. It is obtained by stacking three velocity transformations: one mapping base velocity in the robot frame to object velocity in the object-frame (with the base y-component dropped), and the others mapping gripper velocity in the robot frame to object velocity in the object frame (for Cartesian gripper controllers defined in the robot frame):

$$J = \begin{bmatrix} {}^{o}_{b}R & 0 & {}^{o}_{lg}R & 0 & {}^{o}_{rg}R & 0 \\ {}^{o}_{b}p_y \;\; -{}^{o}_{b}p_x & 1 & {}^{o}_{lg}p_y \;\; -{}^{o}_{lg}p_x & 1 & {}^{o}_{rg}p_y \;\; -{}^{o}_{rg}p_x & 1 \end{bmatrix} \tag{1}$$

where ${}^{B}_{A}R$ denotes a 2×2 rotation from frame A to B, and ${}^{B}_{A}p$ denotes the position of the origin of frame A in B. Subscripts indicate the appropriate frame, with the robot base b, object o, left-gripper lg, and right-gripper rg. To use Eq. 1 for control, the simplest method is to drop the base-y component (second column of J) to obtain the controllable Jacobian $\bar{J}$ and apply the pseudo-inverse to obtain body velocities from desired object velocities:

$$ {}^{r}_{o}v = \bar{J}^{+}\,{}^{o}_{o}v \tag{2}$$

[2]The base has 3 effective DOF, although only 2 are directly controllable.

[3]Force and torque is often difficult or impossible to control precisely; Many arms are not equipped with joint-torque sensors, and even given a force-torque sensor at the gripper, closed-loop force-control can be noisy and error-prone without accurate models of the dynamics of the manipulator.

[4]The mapping from planar gripper velocities to manipulator joint velocities was achieved using the manipulator Jacobian.

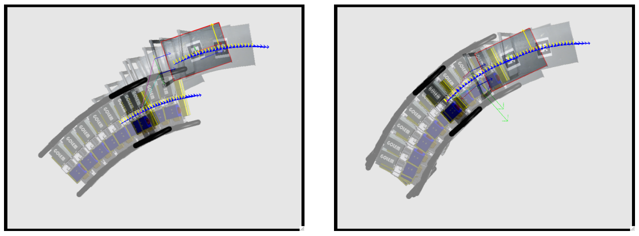

(a) Without Steering (Infeasible) (b) With Steering

Fig. 2. Comparison of body Jacobian with and without steering for a cart pushing trajectory.

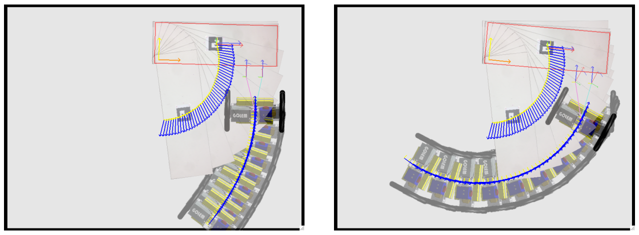

(a) Without Steering (Infeasible) (b) With Steering

Fig. 3. Comparison of body Jacobian with and without steering for a table pulling trajectory.

By simulating the outcome of this controller in the *PBRL* world corresponding to the robot's current beliefs, we can generate trajectories which are expected to succeed in the real world.

The Jacobian-based control approach described above offers a sound method for planning robot trajectories that are accurate and safe to execute, but we found that it frequently allowed the arms to drift out of the reachable workspace (Fig. 2(a) and Fig. 3(a)). This problem comes from having a differential drive base: any desired object velocity perpendicular to the wheels could only be achieved with the arms. We addressed this problem by splitting the Jacobian transform into two parts, and running a steering controller for the base at each intermediate waypoint before simulating the gripper velocities.

The process for each time-step can be summarized as follows:

1) Integrate the desired object-frame velocity ${}^{o}_{o}v_i$ to obtain an ideal object waypoint, assuming no constraints.
2) Integrate the corresponding robot base velocity ${}^{r}_{r}v_i$ to compute a desired base waypoint, and then call a base steering controller to obtain a reachable base waypoint.
3) Compute the residual error between the updated object location given the new base pose and the waypoint computed in step 1 (*i.e.* the work to do be done by the arms), and apply a manipulation controller to minimize this error.

The base controller computes wheel velocities to achieve the desired base velocity, and steering can be achieved by adding a term to produce an angular velocity that turns the robot to minimize the y (lateral) velocity. Incorporating the steering controller into the trajectory generation process relieved the burden on the arms to control errors in the y direction, allowing the grippers to remain in the middle of their workspace. Fig. 2 and Fig. 3 illustrate the effect of adding steering to the body controller.

In summary, the manipulation controller is a mapping from desired object velocities ${}^{o}_{o}v \in \mathbb{R}^3$ to satisfying in-plane body velocities $[{}^{b}_{b}v, {}^{b}_{lg}v, {}^{b}_{rg}v]^T \in \mathbb{R}^8$. The gripper velocities can then be mapped to joint velocities $\dot{q}_r, \dot{q}_l \in \mathbb{R}^7$ by the manipulator jacobians, subject to the constraint that out-of-plane translational and rotational velocities are zero. Overall this formulation reduces the dimensionality of the $3+7+7 = 17$ DOF system to a 3-dimensional space.

B. Planning

Using the method described above, the only free parameters for manipulation are (1) the grasp point on the object (2) the desired object velocity, and (3) the duration of the action. We can compactly parameterize the grasp space as an angle at the object center-of-mass. Candidate grasp points are obtained by computing the boundary point at this angle, projecting the boundary point off the object face (to leave room for the arms to grasp the object), and checking for a collision-free navigation path to this point. If a path is found, this point is added to a list of viable grasp points. This process can be visualized in Fig. 4. Manipulation planning therefore consists of searching the space of valid grasp angles, object velocities, and trajectory lengths, and evaluating the outcomes in the simulator.

Solving a NAMO subtask involves running this manipulation planner to attempt to find a way to clear an obstacle between two given free-space regions. To solve the subtask we perform monte-carlo sampling of model beliefs, and for each model we perform a search over manipulation parameters $a_m \in \mathbb{R}^5$ to generate candidate manipulation trajectories.

Recall that the reward function R^a_s for this subtask is the sum of two terms: the average cost of the manipulation trajectory plus a discounted reward for the region reached. We chose a cost function to penalize total time, force, and torque applied to the object. We achieve this with a decreasing sigmoid function that is shifted and scaled to the range $y \in [0, \alpha]$ for total time t_{total}, total linear force f_{total}, and total torque τ_{total} applied during a manipulation action:

$$R^a_s = \frac{1}{3}\sum_{i=1}^{3} \frac{\alpha}{1+\exp(\beta(\frac{x_i}{m_i} - \eta))} - \alpha + V_{s'} \quad (3)$$

for quantities $x = [t_{total}, f_{total}, \tau_{total}]$ with nominal maxima $m = [t_{max}, f_{max}, \tau_{max}]$. The shape parameters α and β control the magnitude and steepness, respectively, of the cost function. In the experiments in Section VI we set $\alpha = \frac{200}{4} = 25$ (for terminal reward 200), $\beta = 10$, and $\eta = 5$.

High-level NAMO planning proceeds by sequentially solving these subtasks in an ordering imposed by the current free-space region values. This causes the overall planning process to proceed backwards from the goal as free-space values are computed and propagated to neighboring regions. For further discussion of this effect, and demonstration (in simulation) on larger domains, see [7], [8].

C. Manipulation Policies

The inner-loop of the planning process described above requires integrating the body velocities in a simulator parameterized by robot's current world beliefs to obtain executable trajectories. This process can be computationally expensive, particularly if we wish to perform many Monte-Carlo samples to obtain high confidence.

For this reason, we define a *model-dependent manipulation policy*, which constrained the set of achievable velocities and grasp points as a function of object dynamics:

$$a_m \sim \pi(s;\phi) \tag{4}$$

Where ϕ denotes the model parameters. The form of π can be arbitrary, and in this work we consider a (hand-defined) decision tree over ϕ with k dynamics categories. Each leaf node corresponds to a single model class, and each class parameterizes a distribution over action parameters $a_m \in \mathbb{R}^5$. Rather than uniform random search over a_m, we sample actions from π while building our search tree. The policy we describe can thus be viewed as an optional planning heuristic which trades completeness for computation time.

V. Implementation

A. Model Parameterization

Our focus is on manipulating large indoor objects, such as tables and chairs. We found that a single constraint class, anisotropic friction, was very expressive, and captured all of the effects of interest in our office setting. We review the anisotropic constraint model from [14] here, and the requirements for accurately estimating its parameters (along with object mass) from data.

Anisotropic friction is a velocity constraint that allows separate friction coefficients in the x and y directions, typically with one significantly larger than the other. In this work we parameterize the anisotropic friction joint by the 5-vector $J_w = \langle w_x, w_y, \mu_x, \mu_y, \mu_\theta \rangle$, corresponding to the joint position in the body frame, the two orthogonal friction coefficients, and an angular friction term. The angular term was added from the presentation in [13] to capture the wheel drag when rotating an about about a fixed point. In addition to simulating wheels, this constraint can also model planar revolute motion such as a locked caster (by making both linear coefficients large), or fully static (by making all 3 coefficients large).[5]

B. Policy Definition

We consider four model classes for the purposes of defining manipulation policies: *static*, *unconstrained*, *anisotropic*, and *fixed-point*. Static objects have sufficiently large values for all friction coefficients that no manipulation is possible. If an object is known to be static, the manipulation policy is null. Unconstrained objects have no physical constraints, and are free to move in any direction. For unconstrained objects we consider the three planar DOF separately for a total of six directions: ${}^{o}_{o}v \in \{\pm v_x\} \times \{\pm v_y\} \times \{\pm v_\theta\}$. Anisotropic objects have a single large friction coefficient $\mu_x|\mu_y > 0.3$, and behave like wheeled bodies. To manipulate these objects we only consider angular velocities at the constraint point and linear velocities along the unconstrained axis (*e.g.* if $\mu_x < 0.3$ and $\mu_y > 0.3$ then we only consider linear velocities along the constraint's x-axis). Fixed-point objects have large values for both linear friction coefficients $\mu_x, \mu_y > 0.3$, and rotate about the joint anchor. These objects are the most constrained, and to manipulate them we consider only angular velocities $\pm v_\theta$ at the constraint point. Note that this pure angular velocity at the constraint point will produce time-varying linear components at the object center-of-mass, and encoding this transform in the manipulation policy is an additional form of physical domain knowledge that leverages our model representation.

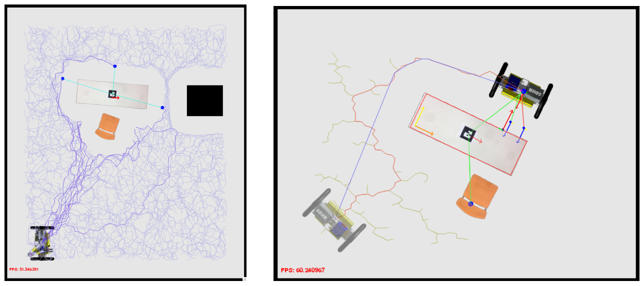

(a) Sampling for Unconstrained Table (b) After Grasping Locked Table

Fig. 4. Sampling reachable grasp points depends on object dynamics. (a) For unconstrained object, points are sampled for each face and projected off the surface to leave room for grasp. (b) After grasping a *fixed-point* object. Note sampled boundary point and gripper offsets.

The grasp angle is chosen to produce grasp points that are manipulable for the target object. If the object is unconstrained, as in Fig. 4(a), the grasp points are sampled from all object faces. If the object has a locked wheel, as in Fig. 4(b), the grasp angles are constrained to the top and bottom face along the opposite end of the object.[6]

These rules were sufficient for our purposes, but learning model-dependent policies is an interesting area of research that may further justify our choice of model representation over non-physics-based alternatives.

C. Predictive Hierarchical Execution

To solve a NAMO problem in realistic environments, the next step is designing a robust method for executing these manipulation actions on a physical robot, and handling when the robot's beliefs are wrong. For NAMO tasks we have three primary considerations:

1) NAMO requires sequencing multiple object interactions, and we therefore must consider navigating to, grasping, and releasing objects, in addition to actual object manipulation.

[5]Note this is different than simply assuming the object has a large mass, because friction can dissipate the energy provided by the robot, whereas a mass-only model does not, resulting in slow drift.

[6]Formally, we selected angles from the set $\pm atan2(\frac{\sqrt{3}}{2}, \frac{w}{2h})$, where w, h denote object width and height, respectively.

Subtask	Parameters
x, y = GetGraspPoint(o_i)	o_i: Target object id
NavToPt(x, y)	x, y: Navigation goal point
Grasp(o_i)	o_i: Target object id
Clear(f_i)	f_i: Target free-space region id
UpdateModel(D)	D: Data gathered during Clear
Release(o_i)	o_i: Target object id

Fig. 5. Subtasks involved in the execution of a NAMO action.

2) The termination condition for manipulation actions is not a specific goal configuration, but rather *any* state that creates an opening.
3) Models may be wrong, so the robot should maintain expectations about the results of its actions, and abort to update its world model as necessary.

For these reasons a single high-level NAMO action is actually composed of a set of distinct intermediate subtasks: finding a valid grasp point according to the world state and model (Section IV-C), navigating to the grasp point, grasping the object, and executing a manipulation trajectory while (a) comparing forces with expected values from planning, aborting and updating the model as necessary, and (b) periodically checking for openings created using a path planner.

To handle this we implement a (stochastic) state machine for coordinating the execution of these operations. An abstract NAMO action instantiates each of the operations above with the appropriate parameters, as defined in Table V-C.

The most important aspect of this construction is the mechanism for execution monitoring and model updating. We achieve this ability by leveraging our physics-engine framework to compute and save gripper forces during manipulation planning.[7] If unexpected forces are observed during execution, the subtask pauses and the data gathered during the aborted episode are passed to a learning procedure based on the method discussed in Section III. After the model has been updated, the planner releases the object and recomputes the NAMO policy for the current world state (note that the aborted action may have changed the action cost, and possibly even the free-space connectivity).

The *Clear* subtask uses the body controller defined in Section IV-A for object manipulation. During manipulation, this subtask periodically checks for openings to the target free-space region using a fast circular-footprint RRT path planner. For each timestep, *Clear* also computes the object-frame forces from the wrist sensor signals, and compares them to the expected forces applied during planning. This subtask terminates when either (a) a force is perceived that exceeds the maximum expected value by a user-defined threshold (150% in our implementation), or (b) the controller stalls or reaches a timeout.

VI. Evaluation

We performed three experiments from identical scene configurations with differing object dynamics in order to highlight our method's robustness to model uncertainty. In addition to replicating the model-identification results from [13], these experiments also show how manipulation control can leverage the learned parameters explicitly for control. These results constitute the first real-robot implementation of the NAMO MDP.

Task configurations were chosen to replicate typical office environments. We selected two tables to use as obstacles in this task: one which is square with actual mass $33Kg$, and another which is rectangular with mass $38Kg$. For these experiments we used a six-camera overhead vision system that tracked the robot and objects using AR-tags [4]. Both tables have four lockable casters which significantly affect the table's dynamics, but their true state is not available to the robot.

Key-frames from an experiment involving two constrained objects are shown in Fig. 6. Planning and execution in this task proceeds as follows: The robot identified two free space regions and selected the square table as the lowest-cost obstacle to clear. When the robot grasped and attempted to push this object out of the way it encountered unexpectedly large force along the pushing direction, and halted to update its model. Using the method from Section III-A, the robot estimated an anisotropic friction constraint in the middle of the table with three large coefficients, effectively rendering it static. The robot then recomputed the object Q-values and elected to pull the rectangular table down to create an opening. Instead of succeeding on its first attempt, the table rotates unexpectedly, due to a locked wheel on the left corner of the table. Rather than aborting, the robot the successfully estimates the coefficients and pose of this constraint, and executes a parameterized fixed-point action to rotate the table about this point (Section IV-C). The result of this rotation action can be seen in Fig. 6(j). This action successfully opened a path to the goal, as can be seen in Fig. 6(k).

This behavior was successfully replicated with the exact same code-base under cases in which both tables were unconstrained (Experiment 2), and in which the square table was static and the rectangular table was unconstrained (Experiment 3). Quantitative results from these experiments can be found in Table 7. Footage from these experiments can be found in the accompanying video, or online.[8]

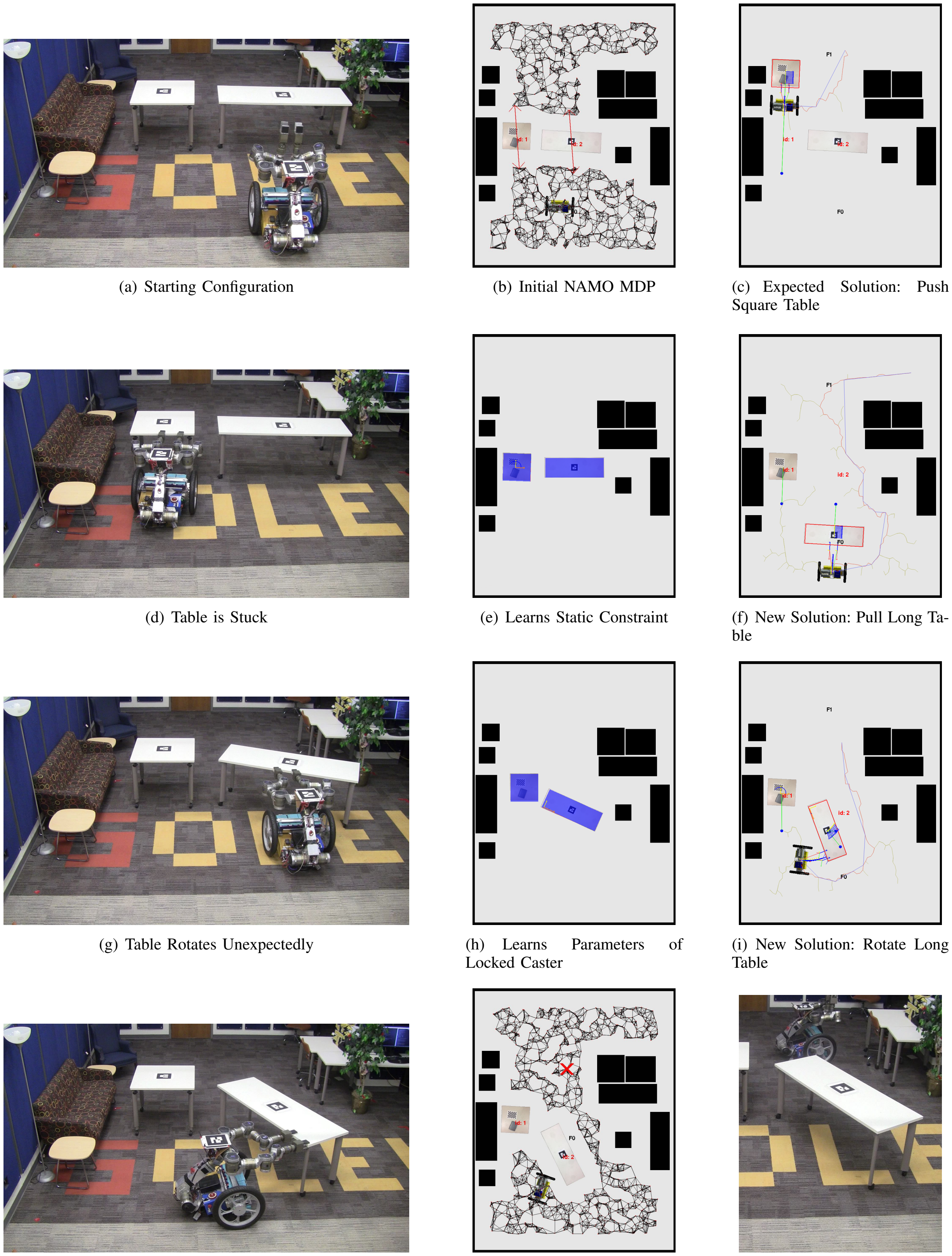
A. Failure Cases

Although the results described here demonstrated adaptive behavior, there were several common points of failure in our experiments. Foremost, many of the subroutines involved in our overall method involved some form of randomness, including PRM free-space detection, RRT path planning, grasp-point sampling, and MCMC model inference. We were able to tune these algorithms to achieve $90+\%$ success rates on each subroutine, but the long horizon of the tasks we consider here led to frequent failures at some point in the pipeline.

In addition, these experiments depended on off-board vision using six overhead cameras, and the limited coverage

[7]This is similar to the idea of efference-copy in biological systems [19].

[8]http://www.cc.gatech.edu/ jscholz6/resources/projects/PBRL/pbrl.php



(a) Starting Configuration

(b) Initial NAMO MDP

(c) Expected Solution: Push Square Table

(d) Table is Stuck

(e) Learns Static Constraint

(f) New Solution: Pull Long Table

(g) Table Rotates Unexpectedly

(h) Learns Parameters of Locked Caster

(i) New Solution: Rotate Long Table

(j) Table Rotated Successfully

(k) Opening Found!

(l) Task Complete

Fig. 6. Key-frames from Navigation Among Movable Obstacles task using physics-based model prior (PBRL). The square table has two locked casters, and the rectangular table has a single locked wheel on the left side. Both of these properties are initially unknown to the robot.

	Experiment 1	**Experiment 2**	**Experiment 3**
Overall Runtime	7m 46s	3m 12s	4m 48s
Overall Planning time	0m 52s	0m 36s	0m 37s
Number of calls to NAMO planner	2	1	1
Number of calls to UpdateModel	2	0	1
Expected action rewards	$183.60, 175.12, 170.37$	173.22	$173.22, 172.88$
Observed action rewards	$-17.10, -18.64, 180.74$	177.15	$-17.48, 176.56$

Fig. 7. Planning and execution statistics for NAMO experiments and selected actions. (1) square-table push, rect-table pull, rect-table rotate, (2) square-table pull, (3) square-table push, rect-table pull. Rewards effectively capture goal bonus (200) discounted by energy consumption and time.

of the system prohibited experiments from any configuration in which the tables were not in the exact middle of the scene.

VII. Discussion

In this paper we presented the first NAMO planner that could adapt to unexpected object dynamics online. This planner made use of a hierarchical MDP planning formalism, and a compact physics-based Reinforcement Learning framework for sample-efficient model estimation. To make planning possible on a real robot with a high-DOF control space, we defined a manipulation controller that reduced the search space to a 5-DOF object-centric representation. We then introduced a model-dependent manipulation policy as a heuristic for searching this space. These controllers and the associated logic for identifying valid grasp points and model parameters were encoded in a state machine for coordinated, robust execution of NAMO obstacle-clearing actions.

Our experiments focused on demonstrating the feasibility of our methods on a manipulation problem that forced the robot to detect errors in its world model and adjust its plan accordingly. The NAMO task presented here included unobservable physical properties, in the form of lockable wheels, that required the robot to either switch objects, or switch control strategies. Our results demonstrated that online physics-based RL is a viable method for solving a mobile manipulation task in a real office environment.

We also demonstrated the value of a compact model parameterization for *planning*, by making direct use of the model parameters as decision variables in a manipulation policy. This would not be possible with a non-inspectable model class, *e.g.* a Gaussian Process. Although it would have been possible for the robot to find a kinematically feasible trajectory through exhaustive search over grasp and velocity parameters, it would require a time-consuming search through a control space $a \in \mathbb{R}^5$, each involving expensive trajectory evaluations. From these observations we would argue that the mapping from model beliefs to control parameters is a useful concept that should be explored in greater depth in the future.

References

[1] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research (IJRR)*, 30(7):954–966, 2011.

[2] Felix Burget, Armin Hornung, and Maren Bennewitz. Whole-body motion planning for manipulation of articulated objects. In *Robotics and Automation (ICRA), IEEE International Conference on*. IEEE, 2013.

[3] M. Deisenroth and C. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[4] Mark Fiala. ARTag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society Conference on*, 2005.

[5] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[6] S.M. LaValle and J.J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research (IJRR)*, 20(5):378–400, 2001.

[7] M. Levihn, J. Scholz, and M. Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Workshop on Algorithmic Foundation of Robotics (WAFR)*, 2012.

[8] M. Levihn, J. Scholz, and M. Stilman. Planning with movable obstacles in continuous environments with uncertain dynamics. In *Robotics and Automation (ICRA), IEEE International Conference on*, May 2013.

[9] Radford M Neal. MCMC using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

[10] Brett Ninness and Soren Henriksen. Bayesian system identification via markov chain monte carlo techniques. *Automatica*, 46(1):40–51, 2010.

[11] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[12] Jonathan Scholz, Sachin Chitta, Bhaskara Marthi, and Maxim Likhachev. Cart pushing with a mobile manipulation system: Towards navigation with moveable objects. In *Robotics and Automation (ICRA), IEEE International Conference on*, 2011.

[13] Jonathan Scholz, Martin Levihn, Charles L Isbell, Henrik Christensen, and Mike Stilman. Learning non-holonomic object models for mobile manipulation. In *Robotics and Automation (ICRA), IEEE International Conference on*, 2015.

[14] Jonathan Scholz, Martin Levihn, Charles L Isbell, and David Wingate. A physics-based model prior for object-oriented mdps. In *International Conference on Machine Learning (ICML)*, 2014.

[15] Mike Stilman, Jon Olson, and William Gloss. Golem krang: Dynamically stable humanoid robot for mobile manipulation. In *Robotics and Automation (ICRA), IEEE International Conference on*, 2010.

[16] Peter Van Overschee and Bart De Moor. N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1):75–93, 1994.

[17] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[18] Gordon Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the fourth annual symposium on Computational geometry*. ACM, 1988.

[19] Daniel M Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural networks*, 11(7):1317–1329, 1998.

[20] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[21] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.