Nonprehensile Whole Arm Rearrangement Planning on Physics Manifolds

Jennifer E. King*, Joshua A. Haustein†, Siddhartha S. Srinivasa*, Tamim Asfour†
*Carnegie Mellon University (CMU), †Karlsruhe Institute of Technology (KIT)
{jeking, ss5}@andrew.cmu.edu, joshua.haustein@student.kit.edu, asfour@kit.edu

***Abstract*—We present a randomized kinodynamic planner that solves rearrangement planning problems. We embed a physics model into the planner to allow reasoning about interaction with objects in the environment. By carefully selecting this model, we are able to reduce our state and action space, gaining tractability in the search. The result is a planner capable of generating trajectories for full arm manipulation and simultaneous object interaction. We demonstrate the ability to solve more rearrangement by pushing tasks than existing primitive based solutions. Finally, we show the plans we generate are feasible for execution on a real robot.**

I. Introduction

In this work, we consider the task of *rearrangement planning* [11, 26, 32, 35]. In these problems, a manipulator must displace one or more objects in order to achieve a goal. Allowing the robot to change the environment makes impossible tasks possible and difficult tasks easier. Consider a robot clearing items off of a cluttered table, such as in Fig.1. Initial solutions to such a problem focused only on using pick-and-place actions [4, 30, 33] to move each object.

More recent solutions to the rearrangement problem use physics based actions, such as pushing, to achieve more efficient solutions [5, 12, 25]. For example, the robot may use one arm to push each item to a location where it can be grasped, lifted and stored using the other arm.

The use of physics models led to far more expressive solutions to rearrangement planning. However, these planners are still bound to the use of high-level primitives, such as straight line pushes. In addition, they require the planner to explicitly reason about moving objects in the scene one at a time, forbidding simultaneous interaction with objects. Finally, they often restrict the interaction to a limited part of the manipulator. For example, only allowing pushing using the end-effector.

These restrictions impose two key limitations on the system. First, the use of motion primitives places a great burden on the designer to develop high-level actions that are both dexterous and efficient, transferring, in some ways, computational complexity on to the designer. Second, reducing the problem to an ordered list of moved objects limits the planner to simply producing combinations of high-level primitives.

We develop a planner that can generate diverse whole arm pushing actions. To do this we remove the reliance on pre-defined motion primitives. We formulate the rearrangement planning problem as a search over the combined state space of the manipulator and objects. Then, we harness the power of a randomized planner, such as the RRT [20], to quickly produce plans in this high-dimensional space. The use of pushing actions introduces non-holonomic constraints in the planning problem. In particular, the motion of the pushed objects is governed by the physics of the contact between manipulator and object. To empower our planner to reason about object motion, we embed a physics model into the core of the randomized planner.

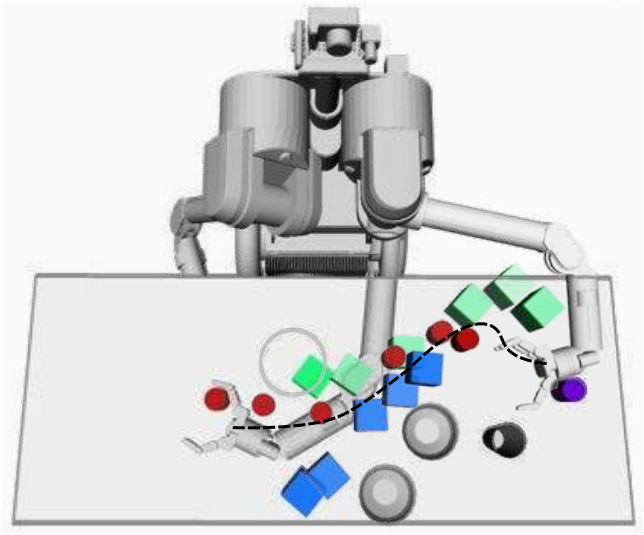


Fig. 1: An example rearrangement plan. The robot is tasked with moving the green box to a pre-defined location where it can be grasped by the right arm. Our algorithm plans an arm trajectory that uses the whole arm to reconfigure clutter such as the blue box , red can and grey bowl out of the way using physics-based nonprehensile actions.

The general solution would utilize a full dynamical physics model, enabling modeling of many types of contact, including pushing, toppling and throwing. While attractive, this solution requires incorporation of velocities into our state, doubling its size. Additionally, it allows the manipulator to make any feasible motion, creating a large action space. The combination of these two attributes makes the search intensive.

In this work, we offer a solution to make solving the rearrangement planning problem with a randomized motion planner tractable. To do this, we use a

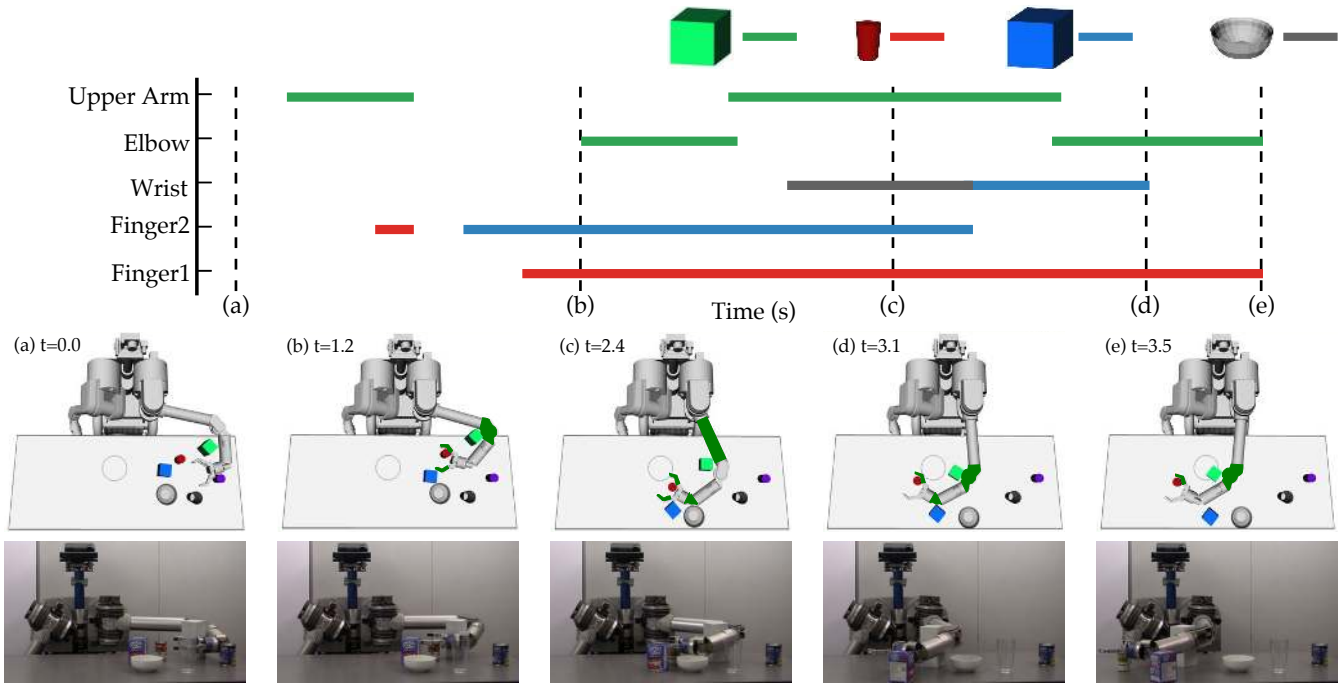


Fig. 2: A solution to the rearrangement planning problem. (Top) Contact between manipulator and objects throughout trajectory. (Middle) Trajectory snapshots at the indicated time points. Link(s) in contact at the time step are highlighted in green. (Bottom) Snapshots of the trajectory execution. As can be seen, the planner generates simultaneous contact with movable objects. Additionally, the full arm is used during rearrangement.

quasistatic model of planar pushing rather than a full dynamical physics model. While this limits our solution space, the quasistatic assumption applies in many manipulation applications [1–3, 8, 10, 24, 27, 36].

Using a quasistatic model of interaction allows us to eliminate the need to consider velocities in our search space, allowing planning in configuration space. Our choice of model also allows us to focus our action selection. In particular, we can project all actions to a constraint manifold parallel to the support surface. This increases the likelihood of an action leading to contact with movable objects in the environment. For example, in the experiment from Fig.1 all actions are projected onto the plane 8cm above the table.

We show that our planner is capable of generating solutions that use whole arm manipulation and allow simultaneous interaction with multiple objects (Fig.2). We demonstrate successful use of our planner on several rearrangement problems.

The remainder of the paper is organized as follows. In Sec.II we formalize the rearrangement planning problem. We explain our method for modeling the problem as a randomized kinodynamic planning problem in Sec.III. We detail our choice of physics model (Sec.III-A), sampling method (Sec.III-B), distance metric (Sec.III-D) and action projection (Sec.III-C). Additionally, we give an initial approach for smoothing generated paths (Sec.IV). We provide several experimental results in Sec.V. In particular, we show that our algorithm is able to solve more problems than existing primitive based solutions. In Sec.VI we discuss the limitations of our algorithm and highlight areas for future work.

II. The Rearrangement Planning Problem

A. Terminology

We use the standard terminology from Lavalle [21]. We work with a robot in a configuration space $q \in \mathcal{C}^R$. Our world is populated with obstacles that must be avoided and also with M movable objects which can be contacted and moved around. Each movable object is endowed with a configuration space $o^i \in \mathcal{C}^i, i = 1 \dots M$. We define the state space X as the cartesian product space of the robot and objects, given by $X = \mathcal{C}^R \times \mathcal{C}^1 \times \cdots \times \mathcal{C}^M$. We denote a state $x \in X$ by $x = (q, o^1, \dots, o^M)$.

Following Lavalle [21] and Simeon et al. [30], we define the free state space $X_{free} \subseteq X$ as the set of all states where the robot and objects are not penetrating themselves, the obstacles or each other. Note that this allows *contact* between entities, which is critical for manipulation.

B. The Problem

Our task is to find a *feasible* path $\tau : [0,1] \rightarrow X_{free}$ starting from a state $\tau(0) \in X_{free}$ and ending in a *goal region* $\tau(1) \in X_G \subseteq X_{free}$.

Feasible plans. The state x evolves nonlinearly based on the physics of the manipulation as its time derivative $\dot{x} = f(x,a)$ where $a \in \mathcal{A}$ is an action from the allowable set of controls that can be applied to the robot. A plan is feasible if there exists a mapping $\pi : [0,1] \rightarrow \mathcal{A}$, such that for all $t \in [0,1,]$, $\dot{\tau}(t) = f(\tau(t), \pi(t))$, i.e. there exist robot actions that can actually perform the plan.

A feasible plan can be quite complicated, making and breaking multiple contacts with movable objects and reconfiguring them to achieve the goal.

Goal region. In manipulation problems, the goal is often underspecified. We define a goal region $X_G \subseteq X_{free}$ as the set of states that achieve a goal criteria.

For example, in [32] only the robot's goal is specified. The final poses of the objects do not matter. To represent this goal region, we can denote the robot's goal as $q_G \in C^R$. Then we define X_G as the set of all states with the robot in configuration q_G. In other problems [13] the task is to move a specific object to a specific place (or a set of places). In these problems, we denote this object as the *goal object* G with its configuration space $g \in C^G$ and its goal as the set $\mathcal{G} \subseteq C^G$. We define $X_G \subseteq X_{free}$ as the set of all states with the goal object in $\mathcal{G}$.

Much of the planning complexity arises because the system is underactuated, with one robot having to move several objects, and nonlinear, due to the physics of manipulation. In Sec.III, we describe how we can combine physics models with ideas from randomized kinodynamic planning to produce fast, feasible plans.

III. Quasistatic Rearrangement Planning for a Manipulator

Algorithm 1 Kinodynamic RRT

```
1: T ← {nodes = {x_0}, edges = ∅}
2: while not ContainsGoal(T) do
3:     x_rand ← SampleConfiguration()
4:     x_near ← Nearest(T, x_rand)
5:     for i = 1...k do
6:         (a_i, d_i) ← SampleAction()
7:         (x_i, d_i) ← ConstrainedProp(x_near, a_i, d_i)
8:     i* = argmin_i Dist(x_i, x_rand)
9:     T.nodes ∪ {x_i*}
10:    T.edges ∪ {((x_near, x_i*), a_i*, d_i*)}
11: path ← ExtractPath(T)
```

We utilize a RRT to solve the rearrangement problem. Traditional implementations of the algorithm solve the two-point boundary value problem during tree extension. Because we must plan in the joint configuration space of the robot and objects, solving the two-point boundary value problem is as difficult as solving the full problem. For example, consider a scene with a single movable object, M^1. To transition from a state $x_1 = (q_1, o_1^1)$ to state $x_2 = (q_2, o_2^1)$, we must first find a collision free path for the manipulator from configuration q_1 to a location near o_1^1, the start location of M^1. Then we must determine an action sequence that pushes M^1 from configuration o_1^1 to configuration o_2^1. Finally, we must generate a collision free sequence of actions to move the manipulator to q_2.

As suggested by Lavalle [22] a useful alternative is to forward propagate all actions under a transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, and select the best using a distance metric defined on the state space. We wish to work with a continuous action space $\mathcal{A}$, rendering full enumeration of the set infeasible. Instead, we approximate it by sampling k actions and associated durations to apply the actions, forward propagating under T and selecting the best from this discrete set. During forward propagation, we apply a physics model to properly capture motion of the manipulator and objects. Alg.1 shows the basic implementation. In the following section, we detail the algorithm.

A. Quasistatic Pushing

We use a quasistatic pushing model with Coulomb friction [23]. In this model, we assume pushing motions are slow enough that inertial forces are negligible. In other words, objects only move when pushed by the manipulator. Objects stop immediately when forces cease to be imparted on the object.

We assume friction between the object and the underlying surface is finite, the pressure distribution between the object and the surface is known and finite, and friction between the object and the manipulator is known. Under these assumptions, we can analytically derive the nonlinear motion of an object when pushed by the manipulator [15, 18]. We allow only manipulator-object contact. We model all interactions with the manipulator. We do not model object-object contact.

Using a quasistatic model of interaction allows us to greatly reduce the size of our search space. As mentioned in Sec.I, we can plan in configuration space rather than state space. Additionally, restricting to pushing in the plane allows $C^i = SE(2)$ for all i, i.e. we can represent the configuration of movable objects by (x, y, θ).

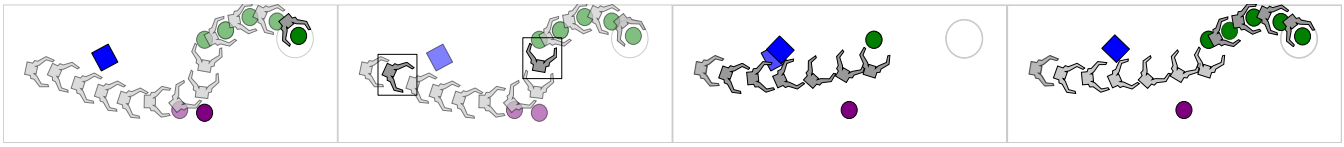
B. Configuration Sampling

We wish to sample a state $x \in \mathcal{X}_{free}$. We can sample from any distribution, as long as we can guarantee that we will densely sample from the space $\mathcal{X}_{free}$. In practice, we sample the robot and all objects from the uniform distribution. We use rejection sampling to ensure the sampled configuration is valid, discarding any sampled states that have object-object collision or manipulator-object penetration.

C. Action Space

We follow the ideas of Simeon et al. [30] and describe feasible plans as an alternate sequence of two types of actions: *transit* and *transfer*. We define *transit* actions as those where the robot moves without pushing any movable objects. *Transfer* actions are those where one or more movable objects are contacted during execution of the action.

Transfer actions are important in rearrangement planning. By definition, the minimal solution to any rearrangement problem contains at least one *transfer*. Our choice of model allows us to focus our action selection



(a) (b) (c) (d)

Fig. 3: The shortcutting algorithm. For simplicity, we depict only the motion of the end-effector. However, the whole arm is considered during planning. (a) The initial path. (b) The two manipulator states selected for connection. (c) The new states generated for the shortcut. (d) The new states added after the shortcut. The motion of the manipulator and green object remain unchanged. However, states are updated to reflect the new location of the blue box and purple circle.

Algorithm 2 The constrained physics propagation function.

Require: A step size Δt

1: **function** ConstrainedProp(x,a,d)
2: $t \leftarrow 0$
3: $q \leftarrow$ `ExtractManipConfiguration(`x`)`
4: **while** $t < d$ **do**
5: $q_{new} \leftarrow$ `Project(`$q + \Delta t a$`)`
6: $a_{new} \leftarrow q_{new} - q$
7: $x_{new} \leftarrow$ `PhysicsPropagate(`x`,` a_{new}`)`
8: **if not** `Valid(`x_{new}`)` **then**
9: **break**
10: $(t, x, q) \leftarrow (t + \Delta t, x_{new}, q_{new})$
11: **return** (x, t)

such that we remain in areas of the robot configuration space where we are likely to generate *transfer* actions. In particular, we project all actions to a constraint manifold parallel to the pushing support surface.

We use the `ConstrainedProp` function shown in Alg.2. This constrained extension behaves similar to that described in Berenson, et al. [7]. During extension, we apply the input action, a, for a small time step, then project the resulting configuration back to our constraint (Alg.2-Line 5). If successful, we generate a new action, a_{new}, that moves directly along the constraint to this projected point (Alg.2-Line 6). This new action is pushed through our physics model (Alg.2-Line 7). The process is repeated for the entire sample duration d or until an invalid state is encountered.

D. Distance Metric

Defining the distance between two states $x, x' \in \mathcal{X}_{free}$ is difficult. Prior work [5] denotes the correct distance metric as the length of the shortest path traveled by the robot that moves each movable object from its configuration in x to its configuration in x'. Computing this distance is intractable. Even approximating this distance is as difficult as solving the rearrangement problem.

Instead, we use a weighted Euclidean metric.

$$Dist(x, x') = w_q \|q - q'\| + \sum_{i=1}^{M} w_i \|o^i - o'^i\|$$

where $x = (q, o^1, \ldots o^M)$, $x' = (q', o'^1, \ldots o'^M)$ and $w_q, w_1, \ldots w_M \in \mathbb{R}$.

The distance metric serves two purposes in the algorithm. First, it is used to define the nearest neighbor in the tree prior to tree extension (Alg.1-Line 4). Second, it is used to select the best control during propagation (Alg.1-Line 8). As stated in Sec.II-B, our goal is underspecified, defining only the location of the goal object. As a result, we assign a higher weight to the goal object in our distance metric. This guides our planner to prefer moving the goal object.

IV. Path Shortcutting

The use of a randomized planner leads to paths that often contain unnecessary movements of the manipulator. Several post-processing techniques to smooth these paths have been introduced in the literature.

One commonly used technique [14, 17, 28, 29] is to randomly select two points from the path and attempt to connect them. If successful, intermediate points between the two selected points are discarded from the path, and the new connection is added.

Use of this technique typically requires solving the two-point boundary value problem to generate the new edge. As stated in Sec.III, this is non-trivial for our problem. Instead we employ a slightly altered technique illustrated in Fig.3.

Given a trajectory produced by our planner, we first randomly select two points in the trajectory (Fig.3b). We then attempt to generate a new action to connect only the robot configuration in these two states. If such an action is generated, we forward propagate it using our physics model, and record the new intermediate states (Fig.3c).

We note that the ending state of the shortcut action could differ from the sampled state in the configuration of one or more movable objects. For example, in Fig.3c the final state differs in the configuration of both the blue box and the purple circle. Because of this, we must forward propagate all remaining actions in the trajectory and ensure the goal is still achieved (Fig.3d). If successful, the updated path is accepted and the algorithm iterates.

V. Experiments and Results

We implement the algorithm by extending the Online Motion Planning Library (OMPL) framework [34].

We test our hypotheses using a dataset consisting of 35 randomly generated scenes. Each scene contains the table as a static obstacle, and between 1 and 7 movable objects in the robot's reachable workspace. Fig.2, Fig.5, andFig.6 show example scenes. We task our

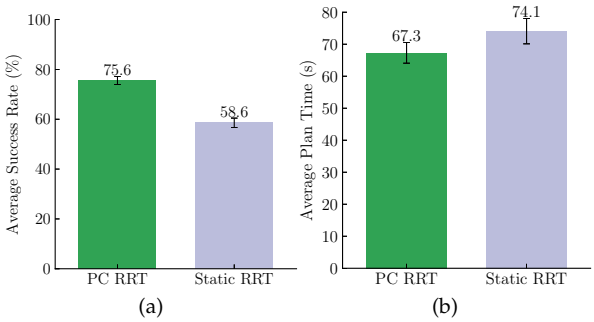


Fig. 4: Comparison of (a) success rate and (b) plan time between our planner (PC RRT - green) and a planner that can only move the target object (Static RRT - purple).

robot HERB [31] with pushing a specified object to a goal region of radius 0.1m using the 7-DOF left arm. In each scene, we use the same goal object. The starting pose of the goal object is randomly selected. The goal region is placed in the same location across all scenes.

We run each experiment 20 times, giving us a total of 700 trials. A trial is considered successful if a solution is found within 300 seconds. We set the number of control samples $k = 3$ for all experiments.

We constrain the end-effector to move in the xy-plane parallel to the table. This allows us to define our action space as the space of feasible velocities for the end-effector. Actions are uniformly sampled from a bounded range. The `Project` function takes the sampled end-effector velocity and generates an updated pose using the Jacobian psuedoinverse:

$$q_{new} = q + \Delta t(J^{\dagger}(q)a + h(q))$$

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

A. Comparison with Baseline Planners

We first compare our planner against two baseline planners. We explore two hypotheses:

H.1 Allowing the planner to move objects via pushing increases success rate and decreases plan time.

H.2 Our algorithm increases success rate and decreases planning time when compared to existing primitive based solutions.

Our first hypothesis **H.1** is motivated by two purposes. First, previous work has demonstrated that allowing the manipulator to move clutter increases the number of problems that can be solved [13, 16]. We verify our planner is consistent with these results. Second, we ensure that the extra time required to propagate with the physics model is not so large that the planner can no longer generate feasible solutions in a reasonable amount of time.

Our second hypothesis **H.2** is motivated by the need to compare our planner against existing state-of-the-art planners that solve the rearrangement planning problem.

1) Planners: We compare our planner against a planner that allows the robot to only push the target object (denoted Static RRT in all results). All movable objects are treated as static obstacles.

Additionally, we compare our algorithm to an implementation of DARRT [5]. Following DARRT, we define three primitives:

1) **Transit** - Move the manipulator from one pose to another via a straight line in configuration space. The motion must be free of collision with any static or movable object in the scene.
2) **RRTTransit** - Move the manipulator from one configuration to another by planning using the RRT-Connect [19] algorithm. The motion must be free of collision with any static or movable object in the scene. The planner is run for 15 seconds before the primitive is considered failed.
3) **Push** - Push (or pull) an object along a straight line from the start pose to the goal pose of the object. The object is caged in the hand during motion, allowing the object motion to be modeled as a rigid connection with the hand. Again the motion must be free of collision with any static objects and all movable objects other than the one being moved.

At each iteration, DARRT chooses to either sample a new pose for the manipulator, a new pose for the target object, or a new pose for a single movable object. In our implementation, we sample these options with equal probability.

If the manipulator is sampled, then the planner attempts to apply the **Transit** or **RRTTransit** primitive. If any of the objects are sampled, the **RRTTransit** primitive is first applied to move the manipulator to the object. Then, the **Push** primitive is applied to relocate the object to its desired position.

2) Statistical Analysis: In all results, we denote our planner Physics Constrained RRT (PC RRT). We run a repeated measures ANOVA using planner (PC RRT, Static RRT, DARRT) as an independent variable. The results show a significant main effect for both success rate ($F(2, 2063) = 204.27, p < 0.0001$) and plan time ($F(2, 2063) = 207.92), p < 0.0001$). Tukey HSD post-hoc analysis reveals all three planners are significantly different from each other using either metric, with p < 0.0001.

Fig.4a compares the average success rate for our planner (PC RRT) with the Static RRT planner across all 700 runs. Fig.4b compares average plan time across all successful runs. Our planner improves success rate by 17% and reduces plan time by 9%.

These results support our first hypothesis: **Allowing the planner to move objects via pushing increases success rate and decreases plan time.**

In Fig.7 we show the expected percent of successful trials for DARRT and our planner given a time budget. Our planner significantly outperforms DARRT for any

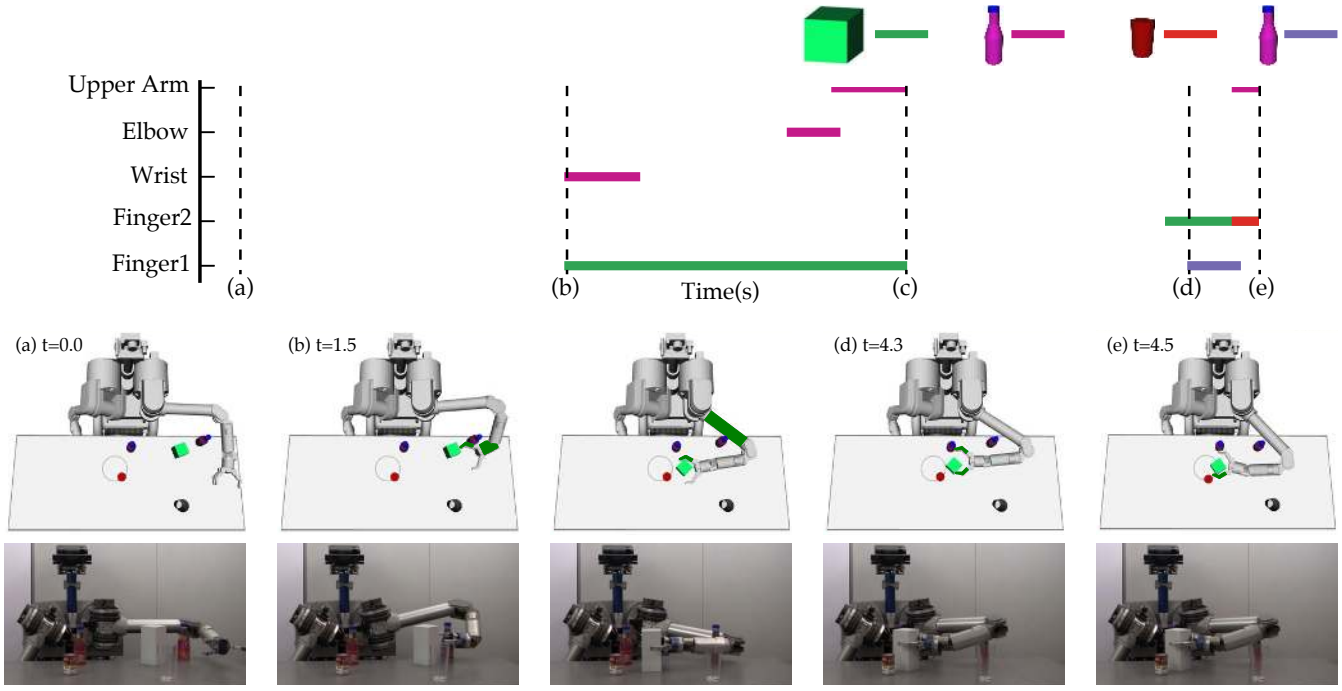


Fig. 5: In this scene, the robot uses the whole arm to manipulate objects in order to achieve the goal. In (b) the robot slides the bottle near the edge of the table in order to make contact with the target object. Then, while pushing the target object into the goal, contact is made between the upper arm and the bottle ((c)). Modeling these contacts allows the robot to ensure the plan will not cause the bottle to fall off the edge of the table.

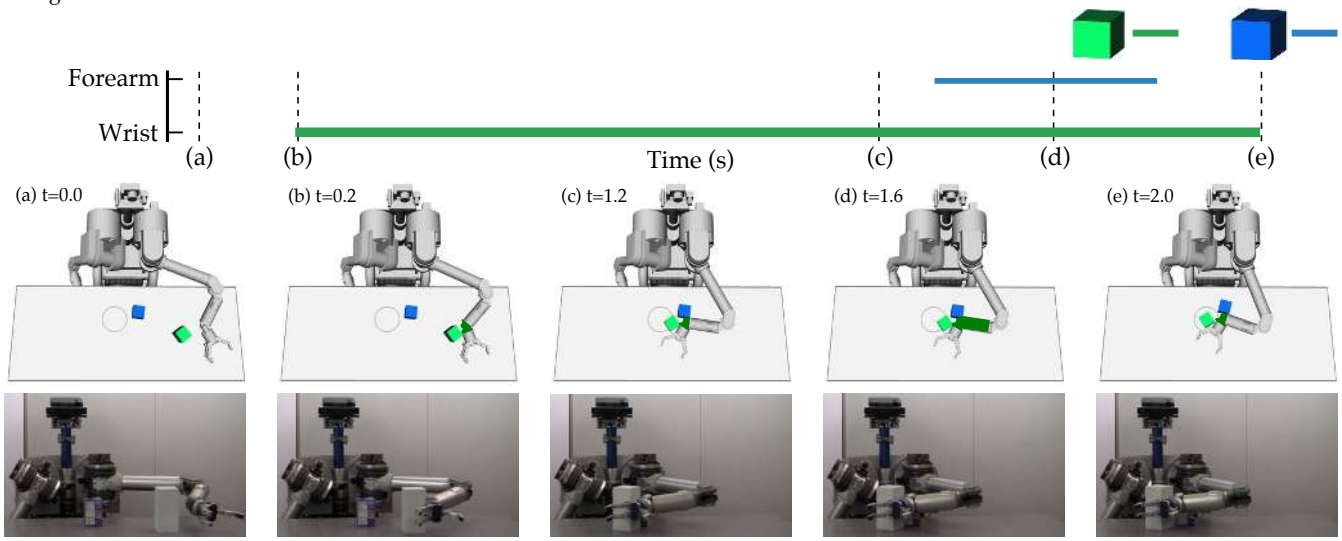


Fig. 6: A low clutter scene. Here the robot uses the forearm and wrist to push the target object into the goal region.

time budget greater than five seconds. For our given time budget of 300 seconds, our planner solves 75% of the test scenes while the DARRT planner is only able to solve 39%.

The results support our second hypothesis: **Our algorithm increases success rate and decreases planning time when compared to existing solutions.**

3) Qualitative Analysis: A deeper look at the solutions our planner achieves provides some insight on the DARRT comparison. Fig.2 shows an example solution for a high clutter scene. The solution found by our planner relies highly on interaction with the full arm. Even in lower clutter scenes such as the one depicted in Fig.5, the planner relies on pushing with multiple links on the manipulator. Additionally, each of the solutions moves multiple objects simultaneously.

Typical primitives-based planners, including our DARRT implementation, allow only interaction with a single object at a time, and restrict that interaction to contact using the end-effector only. This restricts the solutions the planner can consider, causing high rates of failure and longer plan times in scenes with more than just a few items.

However, if we consider scenes with low clutter, as in Fig.6, primitive-based planners such as DARRT work well. In Fig.6, the goal object can be moved directly to

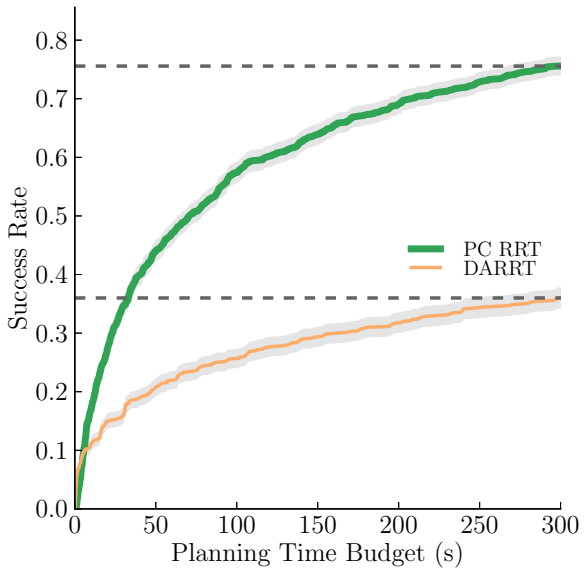


Fig. 7: The success rate of our algorithm compared with the DARRT algorithm. The graph depicts the expected success we can expect each algorithm to achieve given any time budget up to 300 seconds.

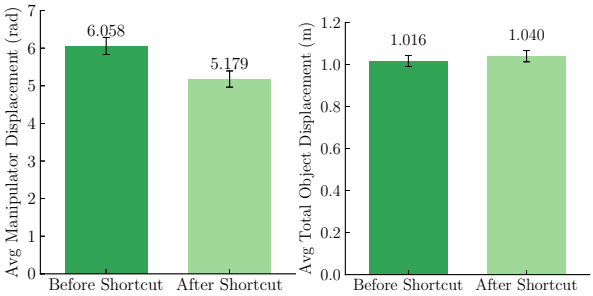


Fig. 8: (a) Manipulator path length before and after path shortcutting. (b) Total object displacement before and after path shortcutting.

the goal region, without the need to clear clutter. The **Push** primitive can solve this trivially, allowing for very low planning times.

This highlights a fundamental tradeoff between our approach and planners similar to DARRT. The effectiveness of primitive based planners is heavily influenced by the richness of the underlying primitive set. For example, were we to augment our DARRT implementation with additional primitives to move the base or perform pick-and-place it is likely the planner performance would improve. Conversely, our planner is able to achieve good performance sampling from a very basic action set, but trades efficiency on scenes that could be solved with a single primitive.

B. Path Shortcutting

In our next results, we examine the performance of the shortcutting algorithm described in Sec.IV. We explore the following hypothesis:

H.3 Applying path shortcutting decreases the length of the manipulator path.

This hypothesis is motivated by the need to check that our shortcut method is able to successfully find segments to remove. We run the shortcut algorithm for a maximum of 15 seconds on each successfully generated path. Fig.8a shows a 14% reduction in manipulator path length after shortcutting.

We run a repeated measures ANOVA for before and after shortcutting and show a significant main effect for manipulator path length ($F(1, 1023) = 11.21, p < 0.001$).

These results support our third hypothesis: **Applying path shortcutting decreases the length of the manipulator path.**

We next analyze the change in total object displacement before and after shortcutting (Fig.8b). This metric measures the combined distance objects are moved in the plan. Interestingly, we see no significant change ($F(1, 1023) = 1.14, p = 0.285$) . This indicates that the segments of the manipulator trajectory being removed during shortcutting are predominantly *transit* segments, i.e. segments where the robot is not interacting with movable objects. This is expected for two reasons. First, the goal of our shortcutting is to decrease manipulator path length while still achieving the goal. The shortcut algorithm is not guided to reduce object displacement. Second, removing *transit* segments has little effect on goal achievement, because we do not remove any interactions with the objects. On the other hand, removing a *transfer* segment could cause overall goal failure if that segment was critical to goal achievement, i.e. we eliminated a push of the goal object. Thus, it is more likely that attempts to shortcut *transfer* segments will be rejected because the goal is no longer achieved.

Scene 1	Scene 2	Scene 3	Scene 4
10/10	4/10	7/10	10/10

TABLE I: Successes for trajectories executed on real robot

C. Real world execution

Finally, we test that the trajectories we generate are able to be executed with some success on real hardware. To do this, we select successful solutions for four different scenes, and run them open loop on the robot. We run 10 trials for each scene and record success or failure based on whether the goal object ended in the goal region. Tab.I shows the result.

We note that with these tests, we are only verifying that our physics model is good enough to achieve some success. We leave full investigation of the accuracy of our model to future work.

VI. Discussion

In this work, we formulate the rearrangement planning problem as a randomized kinodynamic motion planning problem. We show that embedding a physics model into the planner allows us to generate trajectories with full arm manipulation and simultaneous object interaction. Careful selection of this model allows us to reduce our state and action space, making the

search feasible. Our experiments show our solution allows us to solve more problems than primitive based approaches.

Our selection of a planar quasistatic physics model limits us in two ways. First, we can only solve scenes where the quasistatic assumption holds. This prevents us from interactions that require dynamics, such as pushing a ball. Second, we only consider pushing interactions. In future work, we wish to explore the incorporation of full dynamical physics models into our planner. This would allow us to expand the behaviors we generate beyond pushing to actions such as toppling or rolling.

In Sec.V-C we provide some initial results when executing planned trajectories in the real world. While we are able to achieve some success, it is clear that our model is not a perfect description of the physical world. Such a model does not exist. The planner we have presented is just an initial step in incorporating physics models into motion planning. To be truly useful, we must augment the planner to represent and reason about uncertainty. In addition, we must consider ways to incorporate feedback obtained online during plan execution.

We note that while our planner does not require definition of any motion primitives, it is inclusive of them. In particular, if a set of primitives exists, it can be included in the action space. Additionally, our planner can itself be used as a primitive in a hierarchical planner such as [6, 9]. We hope that this can help achieve more expressive solutions to complex manipulation problems.

VII. Acknowledgements

This work is partially supported by the Toyota Motor Corporation, NASA NSTRF Grant NNX13AL61H, the IGEL Program from the College for Gifted Students of the Department of Informatics at Karlsruhe Institute of Technology, the European Union Seventh Framework Programme under grant agreement no. 270273 (Xperience) and the International Center for Advanced Communication Technologies (InterACT).

References

[1] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *IEEE/RSJ IROS*, 1993.

[2] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Sensorless parts orienting with a one-joint manipulator. In *IEEE ICRA*, 1997.

[3] S. Akella and M. T. Mason. Posing polygonal objects in the plane by pushing. In *IEEE ICRA*, 1992.

[4] R. Alami, J. P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *WAFR*, 1994.

[5] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Manipulation with multiple action types. In *ISER*, 2012.

[6] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *IEEE ICRA*, 2013.

[7] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE ICRA*, 2009.

[8] R. C. Brost. Automatic grasp planning in the presence of uncertainty. *IJRR*, 7(1), 1988.

[9] P. C. Chen and Y. K. Hwang. Practical path planning among movable obstacles. In *IEEE ICRA*, 1991.

[10] M. Dogar and S. Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *IEEE/RSJ IROS*, 2010.

[11] M. Dogar and S. Srinivasa. A framework for push-grasping in clutter. In *RSS*, 2011.

[12] M. Dogar and S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 33(3), 2012.

[13] M. R. Dogar, K. Hsiao, M. Ciocarlie, and S. S. Srinivasa. Physics-based grasp planning through clutter. In *RSS*, 2012.

[14] C. V. Geem, T. Siméon, and J.-P. Laumond. Mobility analysis for feasibility studies in cad models of industrial environments. In *IEEE ICRA*, 1999.

[15] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. part 1. limit surface and moment function. *Wear*, 1991.

[16] M. Gupta and G. S. Sukhatme. Using manipulation primitives for brick sorting in clutter. In *IEEE ICRA*, 2012.

[17] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *IEEE ICRA*, 2010.

[18] R. D. Howe and M. R. Cutkosky. Practical force-motion models for sliding manipulation. *IJRR*, 15, 1996.

[19] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE ICRA*, 2000.

[20] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[21] S. M. LaValle. *Planning Algorithms*. 2006.

[22] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *IJRR*, 20(5), 2001.

[23] K. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. In *WAFR*, 1995.

[24] Y. Maeda, H. Kijimoto, Y. Aiyama, and T. Arai. Planning of graspless manipulation by multiple robot fingers. In *IEEE ICRA*, 2001.

[25] T. Mericli, M. Veloso, and H. L. Akin. Achievable push-manipulation for complex passive mobile objects using past experience. In *AAMAS*, 2013.

[26] D. Nieuwenhuisen, A. Stappen., and M. Overmars. An effective framework for path planning amidst movable obstacles. In *WAFR*, 2008.

[27] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars. Path planning for pushing a disk using compliance. In *IEEE/RSJ IROS*, 2005.

[28] G. Sánchez and J.-C. Latombe. On delaying collision checking in prm planning - application to multi-robot coordination. *IJRR*, 21(1), 2002.

[29] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. H. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *IJRR*, 17, 1998.

[30] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *IJRR*, 23(7-8), 2004.

[31] 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*, 28(1), 2010.

[32] M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *IEEE-RAS Humanoids*, 2004.

[33] M. Stilman, J. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE ICRA*, 2007.

[34] I. Sucan, M. Moll, and L. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*, 2012.

[35] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path planning among movable obstacles: a probabilistically complete approach. In *WAFR*, 2008.

[36] N. Zumel and M. Erdmann. Nonprehensile manipulation for orienting parts in the plane. In *IEEE ICRA*, 1997.