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

Rearrangement planning using object-centric and robot-centric action spaces

Jennifer E. King*, Marco Cognetti†, Siddhartha S. Srinivasa*
*Carnegie Mellon University (CMU), †Sapienza University of Rome
{jeking, ss5}@andrew.cmu.edu, cognetti@diag.uniroma1.it

***Abstract*—This paper addresses the problem of rearrangement planning, i.e. to find a feasible trajectory for a robot that must interact with multiple objects in order to achieve a goal. We propose a planner to solve the rearrangement planning problem by considering two different types of actions: *robot-centric* and *object-centric*. *Object-centric* actions guide the planner to perform specific actions on specific objects. *Robot-centric* actions move the robot without object relevant intent, easily allowing simultaneous object contact and whole arm interaction. We formulate a hybrid planner that uses both action types. We evaluate the planner on tasks for a mobile robot and a household manipulator.**

I. Introduction

In this paper, we offer a method for solving rearrangement planning problems. In these problems, a robot must work in clutter, reasoning about moving multiple objects in order to achieve a goal. In particular, we focus on rearrangement planning problems that require nonprehensile interactions, such as pushing, to interact with objects.

Several previous works have solved the problem using purely *object-centric* actions [1]–[5]. Here the planner is guided to perform specific actions on specific objects. These *object-centric* actions are often exposed to the planner in the form of user-defined high-level primitives. These primitives can be highly effective - allowing the planner to make large advancements toward the goal. However, the use of only these actions limits the types of solutions generated by the planner. In particular, these interactions usually involve only contact with a single part of the robot, i.e. the end-effector, and often forbid simultaneous object contact.

Recent work [6] proposed the use of lower-level primitives that describe only robot motions, with no object-relevant intent or explicit object interaction. The use of these *robot-centric* motions relaxes the restrictions imposed by the *object-centric* actions, allowing the planner to generate solutions that exhibit whole arm interaction and simultaneous object contact. However, the resulting planner often suffers from long plan times due to the lack of goal directed motions available to the planner.

Humans use a diverse set of actions when interacting with the world. While many of these actions are *object-centric*, focusing on interacting in a specific way with a single object, other interactions are purely coincidental. These interactions are the unplanned result of a motion with different intent. Consider the example of reaching for a milk jug in the back of a refrigerator. One might first carefully slide the juice jug

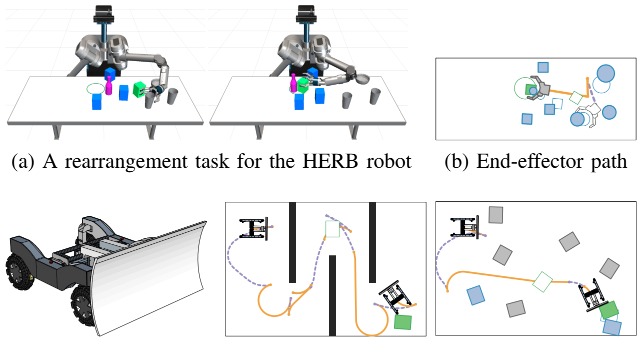

(a) A rearrangement task for the HERB robot (b) End-effector path
(c) The KRex robot (d) Two example paths for KRex

Fig. 1: Rearrangement planning examples for two manipulators, the HERB robot and KRex robot. In each scenario, the robot must push the green box from its start pose to the goal region indicated by the green circle. Our planner combines *Robot-centric* (dashed purple) and *object-centric* (solid orange) actions to generate feasible solutions.

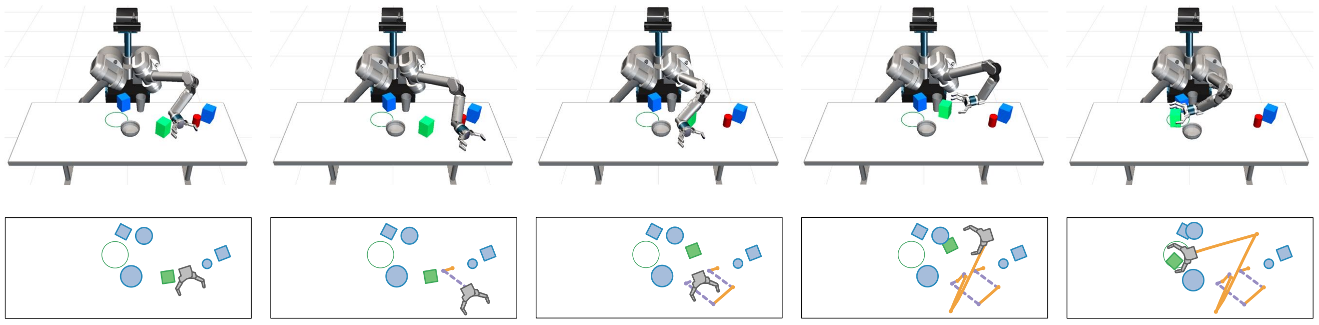
out of the way, then simply reach for the milk, and trust that other objects that are touched will naturally be pushed out of the way, without requiring specific actions to move them. We wish to enable our motion planners to generate similar solutions to planning in clutter.

In this paper, we address the following research question:

> How can we blend the use of *object-centric* actions and *robot-centric* actions to best produce solutions to rearrangement problems?

Our insight is that both types of actions are critical to generating expressive solutions quickly. By integrating the two actions types, we can use the freedom of interaction fundamental to the *robot-centric* actions while still allowing for the goal oriented growth central to the *object-centric* methods.

We propose a planner that incorporates both *object-centric* and *robot-centric* actions. In addition we propose to use physics models like those used in [6]–[8] to forward propagate both the *object-centric* and *robot-centric* actions. This allows the motion primitives central to *object-centric* planners to produce simultaneous object contact and whole arm interaction without explicitly requiring the primitive designer to express and model such interaction. We demonstrate our algorithm on rearrangement planning problems for both a mobile robot KRex and the high degree-of-freedom HERB robot from Fig.1. We show that by allowing the planner to consider hybrid paths



(a) Start configuration

(b) A short transit primitive followed by a random motion

(c) The robot uses the back of the forearm to move the box to a location where a push primitive can be applied.

(d) An *object-centric* push primitive is used to move the box to the goal region.

(e) The final configuration. The robot moves the object, blue box and glass simultaneously to achieve the goal.

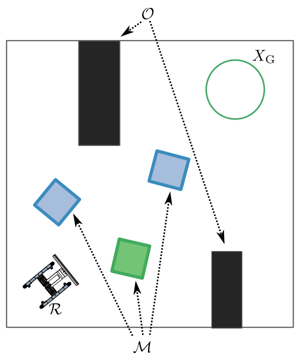
Fig. 2: (Top) The execution of a trajectory that moves the green box from its start configuration to the goal region. The robot uses a mix of random *robot-centric* motions and *object-centric* primitives to accomplish the goal. By using a physics model to rollout all actions the planner can easily generate solutions that use the whole arm and move multiple objects simultaneously. (Bottom) A top-down render of the path of the end-effector. Random *robot-centric* motions are shown in purple, *object-centric* transit and push primitives are shown in orange.

that combine both action types, such as the one in Fig.2, our planner perfoms as well or better than planners that use only *object-centric* or *robot-centric* actions.

The remainder of this paper is organized as follows. In Section II we describe the rearrangement planning problem this paper addresses in detail. We offer an overview of existing solutions and our method in Section III. We then provide experimental evaluation of our planner in Section IV. Finally, we offer limitations and ideas for future extensions in Section V.

II. The Rearrangement Planning Problem

Assume we have a robot, $\mathcal{R}$, endowed with configuration space C^R. The robot is working in a bounded world populated with a set, $\mathcal{M}$, of objects that the robot is allowed to manipulate. Each object is endowed with configuration space C^i for $i = 1 \ldots m$. Additionally, there is a set, $\mathcal{O}$, of obstacles which the robot is forbidden to contact. Fig.3 depicts each of these sets.



$\mathcal{R}$	Robot
C^R	Robot c-space
$\mathcal{M}$	Movable objects
C^i	Movable i c-space
$\mathcal{O}$	Obstacles
X_{free}	Free state space
$\mathcal{U}$	Control space
X_G	Goal region

Fig. 3: The planning environment

We define the state space X as the Cartesian product space of the configuration spaces of the robot and objects: $X = C^R \times C^1 \times \cdots \times C^m$. We define a state $x \in X$ by $x = \left(q, o^1, \ldots, o^m\right), q \in C^R, o^i \in C^i \; \forall i$. 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.

The task of the rearrangement planning problem is to find a feasible trajectory $\xi : \mathbb{R}^{\geq 0} \rightarrow X_{free}$ starting from a state $\xi(0) \in X_{free}$ and ending in a goal region $\xi(T) \in X_G \subseteq X_{free}$ at some time $T \geq 0$.

The state x evolves nonlinearly based on the physics of the manipulation, i.e. the motion of the objects is governed by the contact between the objects and the manipulator. We describe this evolution as a non-holonomic constraint:

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

where $u \in \mathcal{U}$ is an instantaneous control input. The function f encodes the physics of the environment. A path ξ is feasible if there exists a control, $u \in \mathcal{U}$, at every time $t \geq 0$ that allows the constraint f to be satisfied while following ξ.

III. Randomized Rearrangement Planning in Deterministic Environments

We utilize a Rapidly Exploring Random Tree (RRT) [9] to solve the rearrangement problem. Traditional implementations of the algorithm solve the two-point boundary value problem (BVP) during tree extension. Because we must plan in the joint configuration space of the robot and objects, solving the two-point BVP is as difficult as solving the full problem. In particular, the objects are not directly controllable. An object can only move as a result of contact between the manipulator and the object. Therefore, solving the two-point BVP to connect $x_1, x_2 \in X_{free}$ requires finding a path for the robot that moves each object in $\mathcal{M}$ from its position in x_1 to its position in x_2.

Previous works have handled this limitation in two ways, described in the following sections.

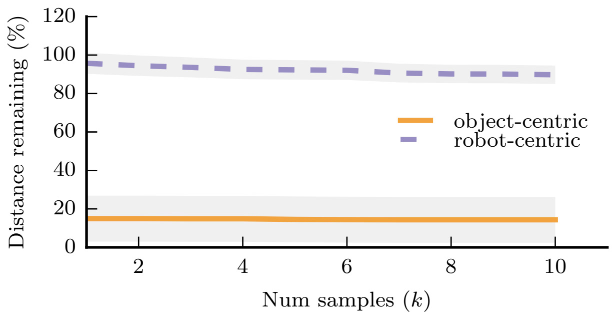


Fig. 4: The distance between the achieved state and targeted state. As can be seen, sampling object-centric actions leads to significant improvement in reaching the target state.

A. *Random sampling*

Algorithm 1 Kinodynamic RRT with random action sampling and physics model propagation

```
1:  $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2:  while not ContainsGoal($T$) do
3:      $x_{rand} \leftarrow$ SampleConfiguration()
4:      $x_{near} \leftarrow$ Nearest($T$,$x_{rand}$)
5:      for $i = 1 \dots k$ do
6:          $(u_i, d_i) \leftarrow$ SampleUniformAction()
7:          $(x_i, d_i) \leftarrow$ PhysicsPropagate($x_{near}, u_i, d_i$)
8:      $i^* = \text{argmin}_i$ Dist($x_i, x_{rand}$)
9:      if Valid(($x_{near}, x_{i^*}$), $u_{i^*}, d_{i^*}$)) then
10:         $T$.nodes $\cup \{x_{i^*}\}$
11:         $T$.edges $\cup \{((x_{near}, x_{i^*}), u_{i^*}, d_{i^*})\}$
12: $path \leftarrow$ ExtractPath($T$)
```

As suggested by Lavalle [10] a useful alternative to solving the two-point BVP is to use a discrete time approximation to (1) to forward propagate all controls and select the best using a distance metric defined on the state space. In particular, define an action set $\mathcal{A} : \mathcal{U} \times \mathbb{R}^{\geq 0}$ where $a = (u, d) \in \mathcal{A}$ describes a control, u, and associated duration, d, to apply the control. Then, a transition function, $\Gamma : X \times \mathcal{A} \rightarrow X$, is used to approximate the non-holonomic constraint.

Our control space $\mathcal{U}$ is continuous, rendering full enumeration of the action set infeasible. Instead, we can sample k actions, forward propagating each under Γ and selecting the best from this discrete set. Algorithm 1 shows the basic implementation.

This technique relies on *robot-centric* actions and uses the transition function Γ to model the interactions that result from these robot motions. Recent work [6], [8] has shown the use of physics models to implement this transition function. This is attractive because it allows complex interactions like multi-object pushing and whole arm manipulation to evolve naturally. It is also straightforward to implement, requiring only a physics model and a sampling method defined on the action space.

The drawback to this technique is its lack of focused tree growth. In particular, during each extension the tree is not strongly pulled toward the sampled configuration. Fig.4 shows an example of the reduction in distance between the end of an extension and the sampled state as the number of samples actions, k, to select from increases. Even with a large value of k the tree makes very little progress toward the sampled state when using *robot-centric* actions. This is particularly detrimental when the sampled state is a goal state. The result is that the tree must randomly "stumble" on a goal rather than intentionally growing in that direction. This often leads to higher than desired plan times.

B. *High-Level Actions*

Algorithm 2 Kinodynamic RRT using motion primitives

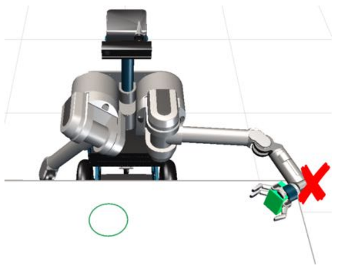
```
1:  $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2:  while not ContainsGoal($T$) do
3:      $x_{rand} \leftarrow$ SampleConfiguration()
4:      $x_{near} \leftarrow$ Nearest($T$,$x_{rand}$)
5:      $a_1, \dots, a_j \leftarrow$ GetPrimitiveSequence()
6:      $x_{new} \leftarrow$ PrimitivePropagate($x_{near}, (a_1 \dots a_j)$)
7:      if Valid(($x_{near}, x_{new}$), $(a_1, \dots, a_j)$) then
8:          $T$.nodes $\cup \{x_{new}\}$
9:          $T$.edges $\cup \{((x_{near}, x_{new}), (a_1, \dots, a_j))\}$
10: $path \leftarrow$ ExtractPath($T$)
```

An alternative method employed in [1], [3] is to use a a set of *object-centric* primitives capable of solving the two-point BVP in a lower dimensional subspace. For example, a "push-object" primitive would be capable of providing a sequence of actions that moves an object from a start configuration to a sampled configuration. *Robot-centric* primitives are restricted to transit motions - actions where the robot moves without making contact with any object in the scene. Algorithm 2 shows the integration of primitives into the RRT.

This method is attractive because it can allow large extensions of the tree, and the sampling method is highly connected to tree growth. Fig.4 shows the reduction in distance to the sampled state on an extension is much better when using these *object-centric* primitives. This is particularly useful when the sample is a goal state: it allows the tree to grow to the goal.

However, the reliance on *object-centric* actions to generate all object motion is detrimental in two ways. First, the actions are limited in their expressiveness. In particular, contact is restricted to only interactions between the manipulator and the single object targeted by the action. The PrimitivePropagate function (Algorithm 2-line 6) explicitly prohibits contact with other movable objects or obstacles in the scene. This prevents simultaneous object interactions, eliminating many feasible solutions when the robot is working in clutter.

Second, and possibly more importantly, this method is susceptible to failure if the primitive cannot be successfully applied. Consider the example in Fig.5a. An example primitive may be to move the hand near the box with palm facing in the direction of the desired push, then push the box in the direction of its sampled location. The box is near the edge of


(a) Desired end-effector pose for a "push-object" primitive is not within the reachable workspace of the robot

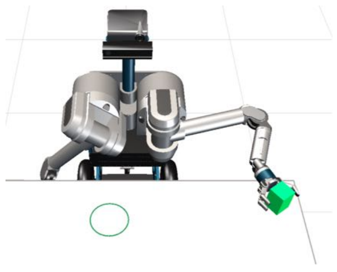

(b) An alternative achievable end-effector pose that cages and pulls the object.

Fig. 5: An example failed "push-object" primitive. The desired end-effector pose is not reachable. An alternate primitive that cages and pulls the object must be defined for the planner to find a solution in *object-centric* primitive based approaches.

the reachable workspace of the manipulator. As a result, all attempts at applying the high-level action will fail because the robot cannot reach the desired pose relative to the box. Even more problematic, a solution to the scene cannot be found given the current action space. To generate a solution, the programmer must define alternative primitives (Fig.5b).

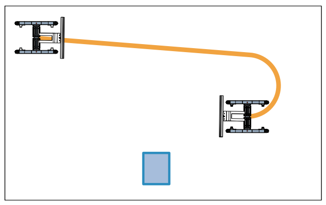
C. Hybrid Approach

Algorithm 3 Kinodynamic RRT using hybrid action sampling

```
1: $T \leftarrow \{\texttt{nodes} = \{x_0\}, \texttt{edges} = \emptyset\}$
2: while not ContainsGoal($T$) do
3:     $x_{rand} \leftarrow$ SampleConfiguration()
4:     $x_{near} \leftarrow$ Nearest($T$, $x_{rand}$)
5:     for $i = 1 \dots k$ do
6:         $r \leftarrow$ Uniform01()
7:         if $r < p_{rand}$ then
8:             $A_i \leftarrow$ SampleUniformAction()
9:         else
10:            $A_i \leftarrow$ SamplePrimitiveSequence()
11:        $(x_i, A_i) \leftarrow$ PhysicsPropagate($x_{near}, A_i$)
12:    $i^* = \text{argmin}_i$ Dist$(x_i, x_{rand})$
13:    if Valid$((x_{near}, x_{i^*}), A_{i^*})$ then
14:        $T$.nodes $\cup \{x_{i^*}\}$
15:        $T$.edges $\cup \{((x_{near}, x_{i^*}), A_{i^*})\}$
16: $path \leftarrow$ ExtractPath($T$)
```

We propose a method that allows for the freedom of interaction fundamental to the *robot-centric* methods while still allowing for the goal oriented growth central to the *object-centric* methods. Algorithm 3 shows the modified algorithm.

Like in the method described in Section III-A, at each tree extension, the best of k possible extensions is selected. However, each candidate extension i expresses a sequence of actions, A_i. With some probability, p_{rand}, the sequence A_i contains a single action $a = (u, d)$ drawn uniformly at random from the space of feasible actions. With probability $1 - p_{rand}$, A_i contains a sequence of actions, $a_1 \dots a_j$, that are equivalent


(a) Example transit primitive

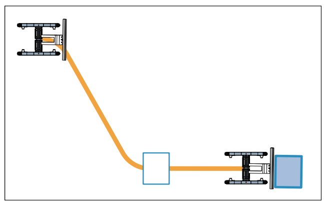

(b) Example push-object primitive

Fig. 6: Two primitives defined for the mobile manipulator. Dubins paths are used to generate paths between two poses in $SE(2)$.

to the primitives described in Section III-B with noise applied to the primitive parameters. In all cases, the sampled action sequence A_i is propagated through the physics model and the sequence is truncated at the first infeasible state encountered, i.e. collision with a static obstacle.

This method is attractive because it combines the strengths of the methods described in Section III-A and Section III-B. Incorporating the physics model in the propagation removes the restriction that *object-centric* primitives can only allow interaction between the manipulator and the object the primitive is defined on. Instead, any unintended contact with other objects in the scene can be modeled. Often, this unintended contact is not detrimental to overall goal achievement and should be allowed. Sampling random actions with some probability allows the planner to generate actions that move an object when all primitives targeted at the object would fail (i.e. the example in Fig.5).

IV. Experiments

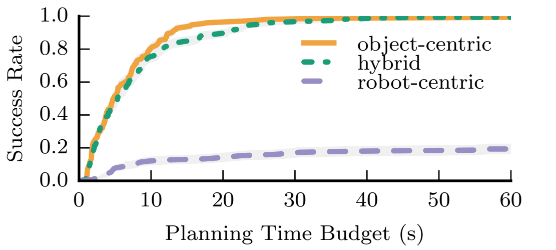
We implement the planner described in Section III-C in the Open Motion Planning Library [11]. We test three versions of the planner. First, we set $p_{rand} = 0$. This forces the planner to always sample primitive sequences. We denote this planner as **object-centric** in all results. Second, we set $p_{rand} = 1$. This forces the planner to always sample a random action. We denote this planner as **robot-centric** in all results. Finally, we set $p_{rand} = 0.5$. This allows the planner to choose primitives or random actions with equal probability. We denote this planner as **hybrid** in all results. We discuss alternative selections for p_{rand} and their impact on results in Section V.

Using the planner, we execute a set of experiments to test the following hypotheses:

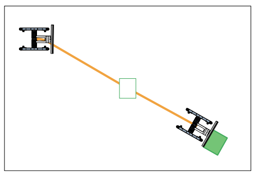
H1: On scenes easily solvable using *object-centric* actions, the **hybrid** planner performs equivalent to the **object-centric** planner in both success rate and plan time. Additionally, both the **object-centric** and **hybrid** planners outperform the **robot-centric** planner that samples only random actions.

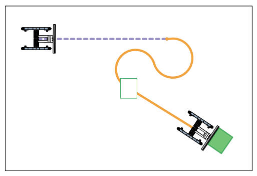
H2: The **hybrid** planner achieves higher success rate and faster plan times than the **object-centric** or **robot-centric** planners on difficult scenes.

We test the hypotheses in two scenarios: a steered car pushing boxes and a 7 degree-of-freedom household manipulator



(a) Success rate as a function of plan time accumulated from 150 runs across three easy scenes.


(b) Example object-centric solution


(c) Example hybrid solution

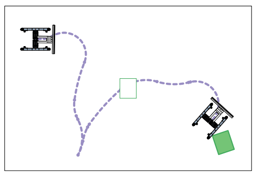

(d) Example robot-centric solution

Fig. 7: Example solutions for each of the three planners for an easy scene.

pushing objects on a table. We define "easy" and "difficult" scenes more precisely in the scenario descriptions.

A. Mobile manipulator

1) Problem setup: We first test our planner using a mobile manipulator (Fig.1c). The robot behaves as a steered car. For this robot, $C^R = SE(2)$ and a control $u = (v, \delta) \in \mathcal{U}$ describes the forward velocity and steering angle applied to the robot. The robot interacts with objects in the plane. We use the Open Dynamics Engine (ODE) [12] as our physics model to forward propagate all actions. We use objects that are quasistatic in nature, i.e. the object comes to rest as soon as the robot ceases to apply forces. This allows us to represent the state of objects by only their configuration, ignoring object velocities. As a result, $C^i = SE(2)$ for $i = 1 \dots m$.

We task the robot with pushing a box from its start configuration into a goal region with radius $0.5m$. We confine the robot to move in a $15m \times 10m$ region. Any actions that move the robot or any object outside of this bounded region are considered invalid.

For this problem the following primitive set is defined:

1) Transit: The transit primitive moves the robot from a start to a goal configuration in $SE(2)$ by finding the shortest length Dubins curves [13] connecting the two configurations (Fig.6a).
2) Push: The push primitive pushes an object along the straight line connecting a start and goal configuration for the object. The primitive returns a set of actions that first move the robot to a position and orientation "behind" the object, then drives the robot straight along the ray, pushing the object (Fig.6b).

When sampling random actions, we sample forward velocity from the range $[-0.5, 0.5]$ m/s and duration from the range 0.5s to 5.0s. We define the sampling range for steering angle using the same minimum and maximum angles as used for generating the Dubins paths.

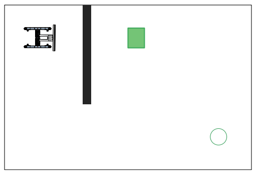
2) Simple scenes: We first test our planner on three scenes solved in less than 10s on average using the *object-centric* primitives defined in the previous section. We denote these three scenes "simple" or "easy" scenes. Fig.7 shows an example scene and a single solution from each planner. We run each planner 50 times on each scene, for a total of 150 trials per planner. For each trial, we record the total time to find a solution. Fig.7a shows the success rate as a function of plan time for up to 60 seconds of total plan time budget. As can be seen, both the **hybrid** and **object-centric** planners solve all scenes in the allotted plan time, while the **robot-centric** planner fails often. A one-way ANOVA with Tukey HSD post-hoc analysis reveals there is no significant difference in mean plan time between the **hybrid** and **object-centric** planners ($p = 0.297$). There is a significant difference between **hybrid** and **robot-centric** ($p < 0.001$) and **object-centric** and **robot-centric** ($p < 0.0001$). This supports our hypothesis: *The **hybrid** planner outperforms the **robot-centric** planner, and performs as well as the **object-centric** planner on simple scenes.*

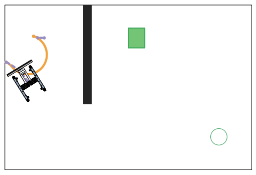
3) Difficult scenes: Next we test our planner on three scenes that require more than 60s on average to be solved with *object-centric* actions. We denote these three scenes as "difficult" scenes. Fig.1d and Fig.8 show these scenes and example solutions. Again we run each planner 50 times on each scene.

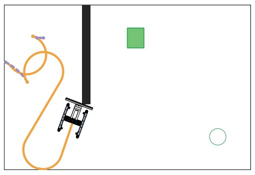
Fig.9 shows the success rate of all three planners as a function of plan time. As can be seen, the **hybrid** planner is far more successful than the **object-centric** or **robot-centric** planners. A one-way ANOVA with Tukey HSD post-hoc analysis reveals that the difference in mean plan-time between the **hybrid** and **object-centric** planners is statistically significant ($p < 0.001$) as is the difference between the **hybrid** and **robot-centric** planners ($p < 0.05$). This supports our hypothesis: *The **hybrid** planner achieves higher success rate and faster plan times than the **object-centric** or **robot-centric** planners on difficult scenes.*

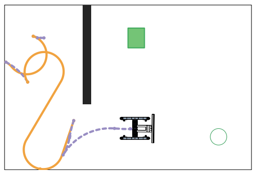
4) Qualitative analysis: Next we examine some qualitative aspects of the solutions. Neither the push primitive nor the transit primitive allow the robot to move in reverse. As a result, in difficult scenes like in Fig.8, we see random actions used to back the robot away from boundaries and obstacles (Fig.8b, Fig.8c).

Additionally, in the difficult scenes (Fig.1d, Fig.8) a static obstacle blocks any path from the start configuration to an object. Thus the push primitive fails in most applications from the start. In addition, the static obstacles also cause many applications of the transit primitive to fail, as they often drive the robot into an obstacle. Here, the use of random


(a) Start configuration


(b) A *robot-centric* random action is used to reverse the robot away from the boundary


(c) A transit primitive moves the robot near the wall


(d) Two *robot-centric* random actions are used to reverse the robot away from the obstacle and then drive the robot into open space

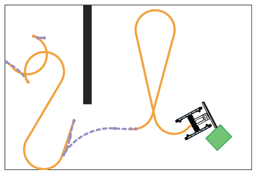

(e) Finally, an *object-centric* action is used to push the target to the goal region.

Fig. 8: In this scene, the wall (black) serves as an obstacle preventing application of the push primitive from the start configuration. *Robot-centric* actions are needed to grow the tree until such a primitive can be applied. Once available, the *object-centric* push primitive extends the tree to the goal.

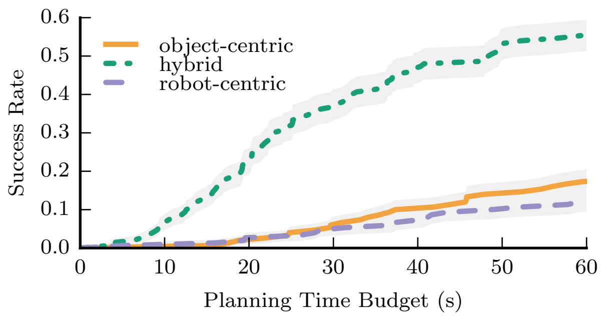


Fig. 9: Success rate as a function of plan time on scenes difficult to solve using only the available primitives (i.e. Fig.1d and Fig.8)

actions available to the hybrid planner is advantageous because they can be used to start tree growth from the root. Then, after the tree begins growing, the push primitive and transit primitive can be applied more freely. In fact, if we analyze all solutions generated by the hybrid planner (across easy and difficult scenes), we see that 89.2% (207/232) end in an object-centric primitive. This supports our intuition that *object-centric* actions help grow the tree to the goal.

B. Household manipulator

1) Problem setup: Next, we execute a series of tests in simulation for the HERB robot [14]. We task HERB to push an object on a table top from a start configuration to a 10 cm radius goal region. We plan for the 7 degree-of-freedom (DOF) left arm of the robot, thus $C^R = \mathbb{R}^7$ and a control for the arm defines joint velocities for each of the 7 DOFs. Following the example in [6], we confine the end-effector of the robot to move in the plane parallel to the table surface. We note that we allow and model pushing contact between the full arm and all objects. Any motions that push an object off of the table are considered invalid.

For this problem the following primitive set is defined:

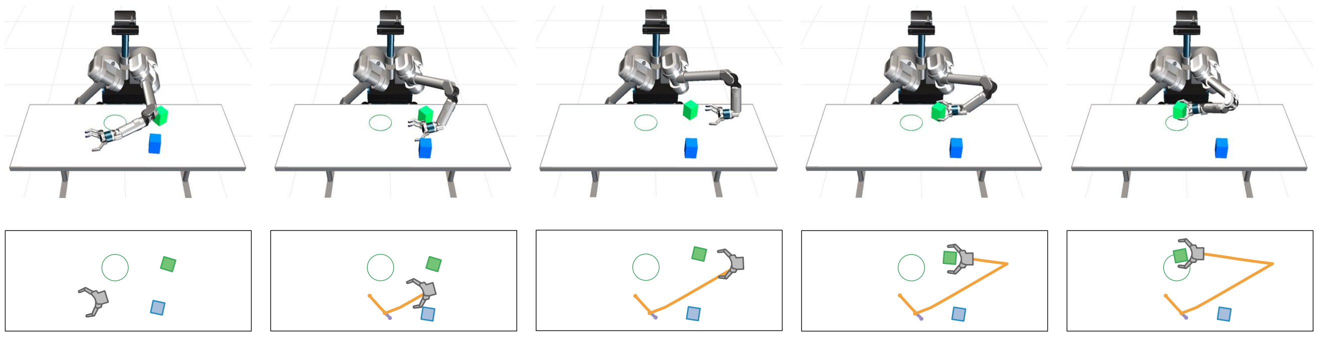
1) Transit: The transit primitive moves the end-effector from a start pose to a goal pose. The motion of the end-effector follows a straight line in workspace along the plane parallel to the table surface.
2) Push: The push primitive pushes an object along the straight line connecting a start and goal configuration for the object. The primitive returns a set of actions that first move the end-effector to a position and orientation "behind" the object, then move the end-effector straight along the ray, pushing the object. The motion of the end-effector is confined to the plane parallel to the table surface during the entire primitive.

We use a quasistatic model of planar pushing as our physics model [15]. Because we only model objects moving in the plane, $C^i = SE(2)$ for $i = 1 \dots m$. We note that more complicated primitives could be defined. For example, we could implement the transit primitive by calling a motion planner for the arm and allowing the end-effector to move out of the plane. The primitives we use are selected to have computational complexity similar to that of sampling random actions, allowing us to more fairly compare plan times between the **object-centric** and **robot-centric** approaches.

2) Simple scenes: We first test our planner on three scenes solved in less than 60s on average using the primitives defined in the previous section. We denote these scenes as "easy" scenes for the household manipulator. Fig.10 shows an example scene and solution. We run each planner 50 times on each scene, for a total of 150 trials per planner. Fig.11 shows the success rate as a function of plan time for up to 300 seconds of total plan time budget. As can be seen, both the **hybrid** and **object-centric** planners perform equivalently. A one-way ANOVA with Tukey HSD post-hoc analysis confirms the **hybrid** and **object-centric** planners to not differ significantly in mean plan time ($p = 0.782$). The **hybrid** and **robot-centric** do show significant difference ($p < 0.0001$) as do the **object-centric** and **robot-centric** ($p < 0.0001$). This further supports **H1**.

3) Difficult scenes: We then test our planner on four scenes for HERB that require more than 300s on average to solve using *object-centric* actions. We denote these scenes as "difficult" scenes. Each either has clutter blocking the path to the goal, or the goal object near the edge of the reachable workspace of the robot. Again we run each planner 50 times on each scene.

Fig.12 shows the success rate of all three planners as a function of plan time. These scenes are difficult and each planner struggles to find a solution. The **hybrid** planner performs slightly better than the other planners but a one-way ANOVA



(a) Start configuration

(b) The *object-centric* push primitive can be applied almost immediately

(c) The push primitive moves the hand behind the object

(d) Then the push primitive moves the hand in the direction of the push

(e) Goal configuration

Fig. 10: (Top) The execution of a trajectory that moves the green box from its start configuration to the goal region. In easy scenes like this, a path can be found by the hybrid planner that uses mostly transit and push primitives. (Bottom) Top down view of the end-effector motion in the resulting path.

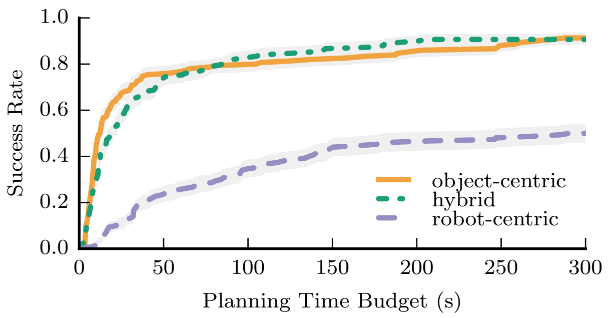


Fig. 11: Success rate as a function of planning time for a set of 50 trials on each of 3 scenes. These scenes can easily be solved using the defined primitives. As can be seen, the hybrid approach performs as well as the *object-centric* approach.

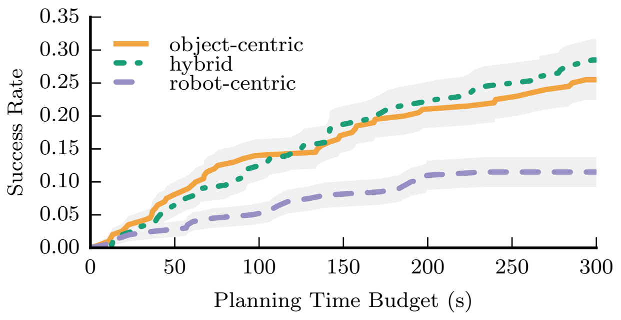


Fig. 12: Success rate as a function of plan time for scenes difficult to solve using *object-centric* primitives

with Tukey HSV post-hoc analysis reveals the difference in mean plan time is not significant when compared to the **object-centric** ($p = 0.629$) or **robot-centric** ($p = 0.566$) planners. This does not support **H2**.

We believe this result is strongly tied to the expressiveness of our primitives enabled by the physics propagation. Specifically, by propagating the primitives through a physics model, we eliminate many of the failure cases that would prevent application of *object-centric* primitives in our scenes, i.e. collision between the robot and other movable objects.

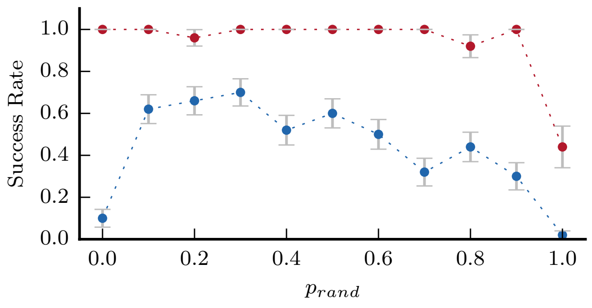


Fig. 14: Success rate as a function of p_{rand} for the scenes in Fig.7 (red) and Fig.8 (blue)

4) *Qualitative analysis:* Similar to the KRex results, we see the random *robot-centric* actions used often to move objects to a location where *object-centric* primitives can be applied. For example, in Fig.2 the push primitive cannot be applied to the object in its start location. The back of the forearm is used to move the box nearer the robot (Fig.2c), enabling the primitive to then be applied (Fig.2d-Fig.2e). Similar to KRex, 80.3% (155 / 193) of paths across all scenes end with an *object-centric* primitive, further supporting our intuition that *object-centric* primitives are important to goal achievement. Fig.1a and Fig.10 show two other example paths ending in *object-centric* primitives.

V. Discussion

In this work, we have presented a method for combining *robot-centric* and *object-centric* action spaces to improve rearrangement planning. Our experiments show that by using a combination of the two action types, we are able to improve success rate and plan time when compared to planners that use only a single action type. In addition, we show that by using a physics model to forward propagate *object-centric* primitives, we are able to allow the primitives to express simultaneous object interaction and whole arm manipulation without explicit encoding.

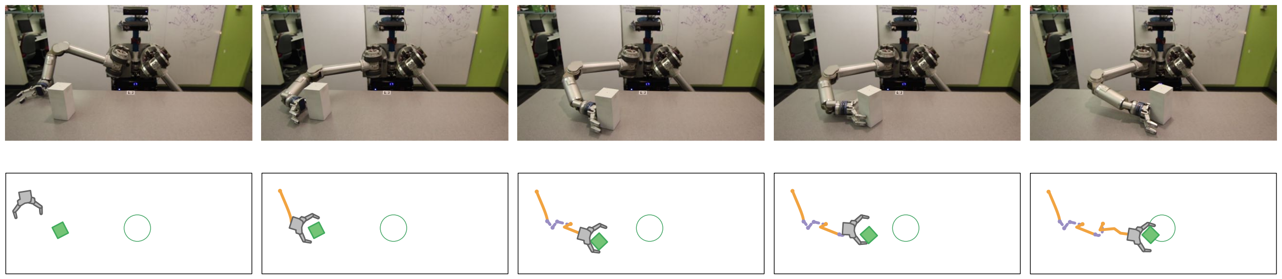


Fig. 13: Successful execution of a planned path containing *object-centric* and *robot-centric* actions on the HERB robot.

In all results, we used a value of $p_{rand} = 0.5$ for our hybrid planner. This value allows *object-centric* primitives and *robot-centric* random actions to be selected with equal probability on each extension. However, we can vary this value to possibly improve performance. Fig.14 shows the success rate as p_{rand} is varied for one of the easy KRex scenes and one of the difficult KRex scenes. The results seem to indicate that low values of p_{rand} are move effective. Future work should examine more closely the relationship between problem composition and p_{rand}.

The randomized nature of the planner means the resulting paths may be highly suboptimal in path length. One common approach to improving optimality is to apply shortcutting techniques to the paths [16]–[19]. These techniques attempt to connect two points on the path with shorter segments that solve the two-point BVP. Applying this technique to rearrangement planning is difficult because the two-point BVP cannot be solved in the general case. However, it may be possible to extend prior work [6] and select points where the two-point BVP only needs to be solved in a lower-dimensional subspace. Then an *object-centric* primitive sequence could be applied.

We focus on scenarios that can be solved using only pushing interactions. Future work should examine the impact of expanding the search to include other interactions such as grasping or toppling. This expanded search increases the size of the state space, requiring objects to be represented in $SE(3)$ rather than $SE(2)$, and the action space, likely requiring we consider actions that move the end-effector in $SE(3)$. However, it may result in simpler plans for scenarios that can easily be solved using alternative interaction primitives, i.e. pick-and-place.

Finally, while we have shown success in executing the generated trajectories on a real robot (Fig.13), some trajectories fail to achieve the goal when executed due to uncertainties in the object perception, robot forward kinematics and physical model of robot-object interaction. Prior work has shown that actions similar to the "push-primitive" used in Section IV-B can be uncertainty reducing [20]. Including such primitives, and guiding the planner to select them, may improve the robustness of the generated trajectories to uncertainty, increasing the planner's applicability in everyday use.

VI. Acknowledgements

This material is based on work supported by the NASA Space Technology Research Fellowship program (award NNX13AL61H), the National Science Foundation IIS (award 1409003), Toyota Motor Engineering and Manufacturing (TEMA), and the Office of Naval Research.

References

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

[2] M. Dogar and S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *AuRo*, vol. 33, no. 3, pp. 217–236, 2012.

[3] S.Jentzsch, A.Gaschler, O.Khatib, and A.Knoll, "MOPL: A multi-modal path planner for generic manipulation tasks," in *IEEE/RSJ IROS*, 2015.

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

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

[6] J. King, J. Haustein, S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *IEEE ICRA*, 2015.

[7] J. Bruce, S. Zickler, M. Licitra, and M. Veloso, "Cmdragons: Dynamic passing and strategy on a champion robot soccer team," in *IEEE ICRA*, 2008.

[8] J. Haustein, J. King, S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable configurations," in *IEEE ICRA*, 2015.

[9] S. LaValle, "Rapidly-exploring Random Trees: A new tool for path planning," 1998.

[10] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *IJRR*, vol. 20, no. 5, pp. 378–400, 2001.

[11] I. Sucan, M. Moll, and L. Kavraki, "The Open Motion Planning Library." *IEEE RAM*, vol. 19, no. 4, pp. 72–82, 2012.

[12] "Open Dynamics Engine," http://www.ode.org, 2000 (accessed August 2014.

[13] L. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[14] 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," *AuRo*, vol. 28, no. 1, pp. 5–20, 2010.

[15] K. Lynch and M. Mason, "Stable pushing: Mechanics, controllability, and planning," in *WAFR*, 1995.

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

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

[18] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning - application to multi-robot coordination," *IJRR*, vol. 21, no. 1, pp. 5–26, 2002.

[19] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. Overmars, "Multilevel path planning for nonholonomic robots using semi-holonomic subsystems," *IJRR*, vol. 17, no. 8, pp. 840–857, 1998.

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