Manipulation with Multiple Action Types

Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, Tomás Lozano-Pérez

Abstract
We present DARRT, a sampling-based algorithm for planning with multiple types of manipulation. Given a robot, a set of movable objects, and a set of actions for manipulating the objects, DARRT returns a sequence of manipulation actions that move the robot and objects from an initial configuration to a final configuration. The manipulation actions may be non-prehensile, meaning that the object is not rigidly attached to the robot, such as push, tilt, or pull. We describe a simple extension to the RRT algorithm to search the combined space of robot and objects and present an implementation of DARRT on the Willow Garage PR2 robot.

1 Introduction

Consider a robot trying to move a plate that is lying flat on a cluttered table to another table. The robot cannot grasp the plate while the plate lies flat on the table, so it has to first maneuver the plate to the edge of the table and then grasp the plate in a way that enables it to later place the plate. This task requires at least three different types of manipulation: push, pick and place.

The classic "pick and place" tasks use only two types of manipulation: transit, in which the robot moves alone, and rigid-transfer, in which the robot moves a rigidly attached object. These two types of manipulation are usually planned separately, connected only by the grasp. Human manipulation, however, is not limited to picking and placing but instead combines many different types of manipulation: pushing,

J. Barry, L. Kaelbling, T. Lozano-Pérez
MIT, Cambridge, MA email:{jbarry, lpk, tlp}@csail.mit.edu

K. Hsiao
Willow Garage, Menlo Park, CA e-mail: hsiao@willowgarage.com

This material is based upon work supported by the National Science Foundation Grant No. 1122374.

pulling, flipping, etc. To extend robotic manipulation capabilities towards those of humans we must be able to plan for and control sequences of diverse actions.

In this paper we outline an algorithm for planning with multiple types of manipulation. The algorithm has the structure of a rapidly exploring random tree (RRT) searching the combined configuration space of the robot and objects. The RRT algorithm attempts to directly connect points in configuration space and, if that fails, samples possible intermediate points and connects those. In many spaces this "connection" is straightforward, usually just a straight line in Euclidean space. Because objects cannot move by themselves, however, such a connection is not viable in the combined space of the robot and objects. Instead, we must use a path that reflects the underlying dynamics of the system. Because of the necessary interactions of the robot and objects during manipulation, this requires modifications to both the extension phase and the sampling phase of the RRT algorithm. We discuss these modifications and describe and analyze experiments run on the PR2 robot.

2 Related Work

There is a large body of work on manipulation actions for us to draw upon. Mason [13] discusses the mechanics of pushing an object, while Brost [2] and Dogar and Srinivasa [5] propose to combine pushing and grasping in a push-grasp, and Huang and Mason investigate striking or tapping objects [9]. However, these papers focus on describing and simulating the dynamics and control of a specific action. When they address planning, they emphasize working with a particular type of manipulation rather than combining it with other types. We take a different view; we assume that by building on this work, we can simulate the forward motion of the object and focus on planning given a diverse set of these actions.

There has also been work on problems that require the robot to manipulate multiple objects. Much of this work assumes only rigid grasping [1, 14, 17, 18, 19] or that each object moves only once [14, 17, 18, 19]. van den Berg et al. [1] relax this second assumption, but their approach relies on describing connected components of a robot's configuration space, which is intractable for high-dimensional configuration spaces. Cosgun et al. [3] discuss trying to place an object on a cluttered surface. They assume only the object to be placed is grasped, but that this object can push other objects out of the way. Multiple objects can be moved at once, but this still incorporates only a single manipulation action. Dogar and Srinivasa [6] consider the problem of trying to move an object in clutter and have a library of manipulation actions, including non-prehensile actions, but assume each object or piece of clutter is moved only once using a single manipulation action. In contrast, we are interested in using multiple types of manipulation to manipulate a single object.

The re-grasping problem [12, 16], especially as framed by Siméon et al., is an example of planning with two manipulation actions. Siméon et al. [16] take a hierarchical approach to the problem, first finding a high-level sequence of transits (motions for the robot alone) and rigid-transfers (motions in which the robot rigidly

grasps an object) and then planning each in the robot's configuration space. Unfortunately, their method relies on the grasped object being able to move instantaneously in any direction, which does not hold for non-prehensile manipulation.

The problem of manipulation with multiple actions is a multi-modal planning problem. Hauser [7] defines a multi-modal planning problem as one in which the system moves among configurations and also among a set of *modes*. The mode space is part of the problem description and each mode describes a set of configurations that all satisfy certain mode-specific constraints. For example, a mode might be the set of configurations in which the robot and object are in a specific grasp. In his initial work Hauser focused on problems with discrete mode spaces, but low-dimensional mode transitions. He showed how to create a two-level roadmap of modes and configurations using interspersed mode and configuration sampling. Later Hauser [8] extended this work to domains like manipulation where the mode space is continuous and used interleaved intra-mode and inter-mode planning to find paths for a walking robot pushing an object on a table. However, that work required the implementation of complicated mode samplers and a number of heuristics, some of which took substantial pre-processing time. Here we show how to solve the problem of manipulation with multiple actions as an RRT with no pre-processing.

3 Problem Definition

We address problems in which we have a robot, a set of movable objects, and a set of diverse, possibly non-prehensile manipulation actions. The input is the configuration space (c-space) of the robot and movable objects, a set of fixed obstacles, a set of manipulation primitives, a starting configuration, and a set of goal configurations. The goal set may be infinite in size. For example, in manipulation, goal positions are often specified only for objects. The goal set is then any configuration in the combined space in which the objects are in their goal positions.

A *manipulation primitive* is a function that takes an initial configuration of the robot and objects and a displacement of the robot's configuration and returns a final configuration of the robot and objects. A *primitive instance* is an instantiation of the primitive with a specific initial configuration and displacement. A *solution* is a sequence of primitive instances that takes the initial configuration into the goal set.

Any type of manipulation can be represented as a manipulation primitive provided it is possible to describe the effect of the primitive on any given configuration of the robot and objects. For complicated primitives it is possible that this would require computational integration of equations of motion, but we use simpler primitives. Throughout this paper we use the following primitives as examples:

- *Transit*: The robot moves alone whenever there is no collision between the robot and the objects or any obstacles in the world.
- *Rigid-transfer*: The robot moves an attached object whenever there is no collision between the robot or object and any obstacles or other objects in the world.

Fig. 1 When the gripper and plate are in two-point contact, the robot can push the plate along the ray connecting its gripper to the plate's center.

- *Pick*: The robot rigidly grasps an object and lifts that object from a support surface when the robot and object are in a feasible grasp configuration.
- *Push*: The robot pushes an object when it has two-point contact between its gripper and the object. The push can only be along the ray connecting the center of the contact points with the center of mass of the object, as shown in Figure 1.

Given a D_R degree-of-freedom robot and n objects each with D_i degrees of freedom, we have a problem with $D_R + \sum_{i=1}^{n} D_i$ degrees of freedom. In Section 4, we discuss our approach to solving this problem.

4 DARRT Algorithm

In Algorithm 1, we present the Diverse Action Rapidly Exploring Random Tree (DARRT) algorithm, a sampling-based algorithm for motion planning problems with diverse, non-prehensile manipulation actions. DARRT has the structure of a rapidly exploring random tree (RRT) with controls [11], but the indirect control of the objects, the high-level manipulation primitives, and the necessity of switching between primitives all require modifications to the classic state sampling, action sampling, and distance metrics. We describe DARRT's EXTENDTOWARDS and SAMPLE methods in detail and discuss its distance metric approximation.

4.1 Extension

We first describe the method for extending from a configuration c_1 towards a configuration c_2. Note that the canonical method of using a short straight line extension in Euclidean space is not applicable here. The Euclidean extension moves c_1 a small amount towards c_2 in each dimension. This moves the robot a short distance from its configuration in c_1 to its configuration in c_2, but also moves each object a short distance from its configuration in c_1 towards its configuration in c_2. Because objects cannot move by themselves, it is not possible to actually execute this extension. We need an extension method that reflects the actual dynamics of the system.

Algorithm 1
Input: M: c-space of movable components, robot R and objects $\{o_1,...,o_n\}$, B: fixed obstacles, A: manipulation primitives, c_I: initial configuration, G: goal set
Output: Graph with a configuration in G.

DARRT(M,B,A,c_I,G)
1 $V \leftarrow \{c_I\}$
2 **while** no configuration in V is in G
3 $\quad s \leftarrow$ SAMPLE(M)
4 $\quad t \leftarrow \arg\min_{v \in V}$ Distance(v,s,M,A)
5 $\quad \{c_1,...,c_l\} \leftarrow$ EXTENDTOWARDS(t,s,M,B,A)
6 $\quad V \leftarrow V \cup \{c_1,...,c_l\}$
7 **return** V

SAMPLE(M)
1 $\{m_1,...,m_j\} \leftarrow$ randomSubset$(\{R,o_1,...,o_n\})$
2 $r \leftarrow$ random configuration for each m_i
3 **return** r

EXTENDTOWARDS(c_1,c_2,M,B,A)
1 $e \leftarrow$ PATH(c_1,c_2,M,A)
2 $\{e_1,...,e_l\} \leftarrow$Discretize$(e)$
3 **for** e_i, **if** collision(e_i,M,B), **return** $\{e_1,...,e_{i-1}\}$
4 **return** $\{e_1,...,e_l\}$

PATH(c_1,c_2,M,A)
1 **if** $c_1 = c_2$ or no useful primitives, **return** $\{\}$
2 $p \leftarrow$ randomUsefulPrimitive(c_1,c_2,M,A)
3 $P \leftarrow$ propagate(p,c_1,c_2,M)
4 $c \leftarrow$ state after applying P to c_1
5 **return** $P \cup$ PATH(c,c_2,M,A)

However, this is not just a case of planning in a non-holonomic system because manipulation usually requires specific relative configurations of the robot and objects. Thus, the subspaces in which primitives can be executed are usually lower-dimensional than the full configuration space. For example, in Pick, the robot must be holding the object in a feasible grasp; for Push, the robot's gripper must be in two-point contact with the object. Both of these primitives require configurations that have zero probability of being sampled at random from the full configuration space. Moreover, non-prehensile manipulation constrains the space in which the object can move. For example, an object that is being pushed can only be moved along a single ray as shown in Figure 1. Thus, even if c_1 is a configuration in which the gripper and object are in two-point contact, the ray along which the object can move must also be "towards" c_2. Therefore, we not only need to pass through configurations in the subspace in which the primitive is executable, we must be in the particular part of that subspace in which the object can be moved towards c_2.

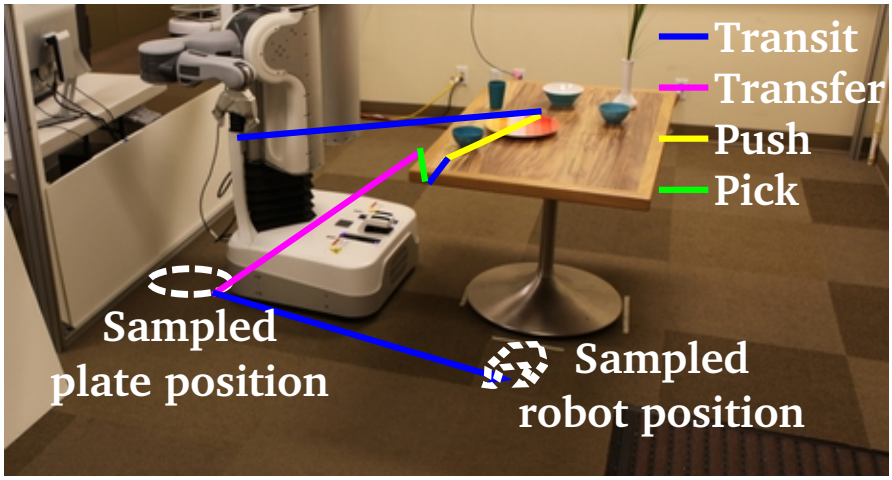
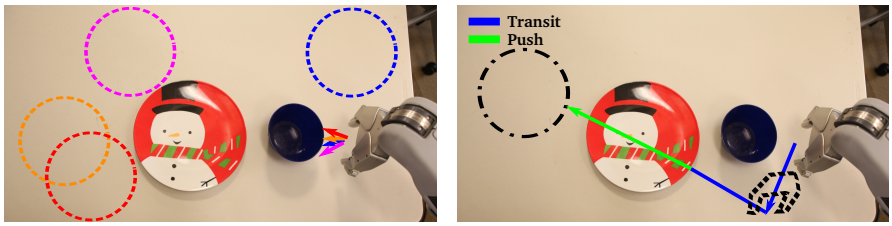


Fig. 2 An extension from the state shown in the photograph towards the sample shown with the white dashed lines. This sequence first transits the robot to a pushing configuration (blue), pushes the plate towards the edge of the table (yellow), transits the robot to a grasp (blue), picks up the plate (green), transfers it to its sampled position (magenta), and finally transits the robot to its sampled position (blue).

The classic method for extending an RRT with actions is to use a single, short application of some action to move a short distance from c_1 towards c_2. However, it is difficult to ensure that such short applications can reach and then remain in the subspaces that must be traversed to reach c_2. Therefore, rather than extend a short way towards c_2, we try to find a sequence of actions that moves all the way from c_1 to c_2. Attempting to extend all the way towards the sampled configuration is the version of the RRT described in Lavalle Chapter 5 (2006).

The problem of manipulation with multiple actions is particularly difficult because the presence of obstacles may require complex paths that use a large number of primitives. Without obstacles, finding a path between two configurations is usually easy. There are a number of ways to implement a search for such a path. In our implementation, we required that a primitive p implement `useful` and `propagate` functions. Propagating c_1 towards c_2 returns a sequence of primitives that applied to c_1 result in a state nearer to c_2 than c_1 is in the subspace in which the primitive operates. A primitive is useful if propagating c_1 towards c_2 will result in a state closer to c_2 than c_1 is. For example, propagating c_1 towards c_2 using Transit results in a state in which the objects are in their positions in c_1 and the robot is in its position in c_2. Therefore Transit is useful if all objects are in the same position in c_1 and c_2 but the robot is not. Push is useful when an object is on a support surface in c_1 and in a different position in c_2. Propagating c_1 towards c_2 using Push returns two primitives: Transit to the pushing configuration and Push from the object's position in c_1 to the object's position in c_2 or the point on the edge of the support surface nearest the object's position in c_2. Pseudo-code is shown in the PATH function in



(a) (b)

Fig. 3 A robot arm pushes a plate on a table. (a) If there are obstacles between the robot and the plate, every direct path (solid colored) will be truncated at the obstacle regardless of the sampled pose of the plate (dashed colored). The colors of the paths added to the tree in this figure correspond to the colors of the sample; i.e. if the plate is sampled in the position shown by the red dashed line, the small red arrow is all that is added to the tree. (b) By sampling robot and object configurations separately, we will eventually sample a configuration for the robot (dashed) that allows a direct path to the plate. A subsequent sample for the plate (dash-dot) results in a much longer extension.

Algorithm 1. Figure 2 shows an example of a path. Note that the path not only includes configurations for executing pushing and grasping, but also ensures that the configuration used for the non-prehensile Push primitive is one that can move the plate towards its sampled position.

Therefore we extend c_1 towards c_2 by finding a sequence of primitive instances that, in the absence of obstacles, takes c_1 to c_2. We check this path for collisions, truncating it to the first collision, and then add the truncated path to the tree.

4.2 Sampling the Space

Although extending with a sequence of actions rather than a single action as described in Section 4.1 finds configurations that lie in lower dimensional subspaces, it creates another problem because we have subspaces with different controllability. We are able to fully control the robot, but can only move objects when the robot can immediately manipulate them. Thus, the robot's first action along an extension will always be to move directly towards a position from which it can manipulate an object. Sampling random configurations from the space does not in fact sample random movements for the robot.

To illustrate, consider a simple world in which a robot arm is pushing a plate on a table as shown in Figure 3. If we simply sample configurations in this world, there is zero probability that we will sample a configuration with the plate in its initial position. Since the plate cannot move on its own, the robot must first move the plate into its new position. A valid pushing path requires that the gripper be in two-point contact with the plate so the first part of the extension will always be a direct robot transit into two-point contact with the plate. If all such direct movements intersect an obstacle, as shown in Figure 3(a), the planning will fail.

To alleviate this problem, rather than sample an entire configuration, we first sample a (possibly proper) subset of the movable components (objects and robot) $O = \{m_1, ..., m_j\}$. We then generate a partially specified sample s in which only the configurations of those components in O are specified. The distance from a fully specified configuration c to s is the distance to the nearest configuration to c such that each of the components in O are in the configurations specified in s. An extension from c to s is any path that results in the components in O being in the configurations specified in s. By using partially specified sample configurations, we allow the robot and objects to take up new positions relative to each other as shown in Figure 3(b).

Sampling partially specified configurations is all that is necessary to reposition the robot and subsets of objects. However, in experimentation, we found that sampling configurations for the robot and objects together tended to be unhelpful because the path is usually truncated before the segment in which the robot moves to its position. Therefore, we either generate a sample that specifies only configurations of objects or one that specifies only a configuration of the robot.

As is common practice when implementing an RRT, some fixed fraction of samples are from the goal set. When sampling from the goal set, we do not explicitly split the sample into robot and object subspaces but we can take advantage of goals for which that split is natural. In many manipulation problems the goal only specifies positions for the objects, in which case a goal sample fits well into this framework.

4.3 Distance Metric

We must also define the distance between two configurations, c_1 and c_2. Because the configuration space contains subspaces that are not directly controllable, the sum of Euclidean distances in each subspace is a significant underestimate of the actual distance between configurations. The correct distance from c_1 to c_2 is the length of the shortest path traveled by the robot that moves each movable component from its position in c_1 to its position in c_2.

Evaluating the correct distance function is, of course, intractable. Thus, we first simplify it by ignoring any obstacles in the world; however, the remaining problem is still hard. The robot must "visit" the action of moving each object exactly once, so this is a version of the Traveling Salesman Problem. In our implementation, we used a greedy algorithm to find the Cartesian distance from c_1 to c_2 by finding a path from c_1 to c_2 in which the robot always moves the closest object first.

5 Results

We implemented DARRT on the Willow Garage PR2 robot and ran experiments in several domains with a variety of manipulation primitives.

5.1 Implementation

We planned for one of the PR2's seven degree-of-freedom arms, its base, and a single rigid object, for a total of sixteen dimensions in the state space. The implementation is built on top of the Open Motion Planning Library (OMPL) [4], which allows the user to define a custom state space, control space, distance metric, sampling algorithm, and extension algorithm for an RRT.

Because we have much finer control over the arms of the robot than we do over the base, we implemented separate transit and rigid-transfer primitives for the arm and the base. At present the planner does not reason about the precision of the primitives, but we hope to add that capability in future work. In total, we implemented eight primitives for the PR2:

- *Arm-Transit*: The arm moves towards a joint goal in a straight line in joint space.
- *Straight-Line-Arm-Transit*: The arm moves the gripper in a straight line in Cartesian space. This was used for approaching and retreating from objects.
- *Arm-Rigid-Transfer:* The arm moves an attached object towards a goal pose using a straight line in joint space.
- *Base-Transit*: The base moves towards a goal pose.
- *Base-Rigid-Transfer*: The base moves an attached object towards a goal pose.
- *Pick*: When the gripper is in a valid grasp pose, the object is attached to the robot's gripper and lifted in a straight line in Cartesian space.
- *Place*: The object attached to the gripper is set down on a support surface in a straight line in Cartesian space and detached from the gripper.
- *Push*: When the gripper contacts the perimeter of a round object on a support surface, the object is pushed along the ray connecting the gripper's center to the object's center. This primitive is shown in Figure 1.

We detected objects at the start of planning using a point cloud from a Microsoft Kinect by segmenting the cloud above the plane of the table. We did not re-detect the objects at any time during execution. We used a three-dimensional map of the environment for collision checking, as shown in Figures 5 and 6. During execution, we used Monte Carlo localization to localize the base of the robot.

5.2 Domains and Problems

We ran experiments on four problems in two different domains, shown in Figures 4-6: Plate1, Plate2, Plate3, and Bowl. In the Plate domain, we had the robot manipulate a flat plate that it could not grasp while the plate was sitting on a table. In the Bowl domain, we demonstrated that the planner works with a bowl that can be directly picked up from a flat surface. In all problems the goal allowed any orientation around the object's z axis, because our objects were symmetric around this axis. To illustrate the difference in execution successes between pushing and rigidly grasping, we did not include the Push primitive in the Bowl domain.

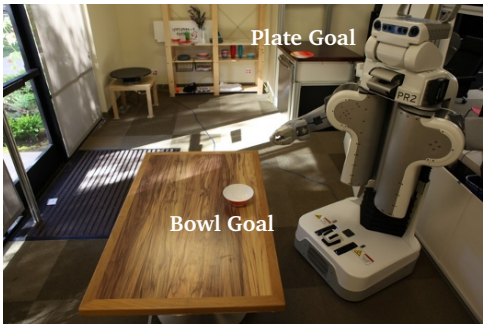


Fig. 4 The world in which we ran DARRT. For each plate problem, the plate started on the central table and the goal was the corner of the side table. The bowl started on the shelf and the goal was on the central table.

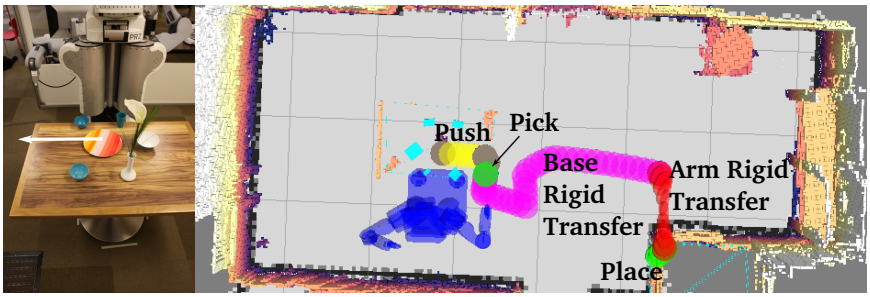
Problem Name	Planning Time (s)	Restart Time (s)	Execution Successes on PR2 Robot
Plate1	257	60	2/5
Plate2	461	200	4/5
Plate3	480	200	3/5
Bowl	66	30	5/5
Plate3 (No Place)	82	60	–
Simple Plate	19	30	–

Table 1 Performance of the unoptimized planner and execution performance on the PR2 for each of the four problems. The Execution Successes are the fraction of the trials in which the robot was able to successfully execute the plan in the real world. We also show planning time for a version of Plate3 in which the goal does not involve a place (Plate 3 (No Place)) and a simple world in which there are no obstacles on the table (Simple Plate). The restart time is the amount of time the planner was given before restarting from the initial state. Times were averaged over 10 trials.

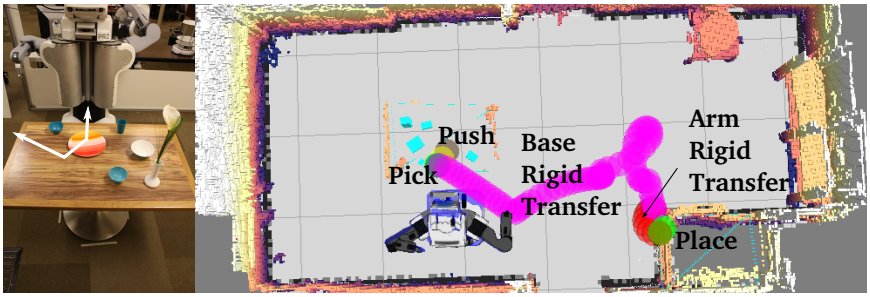
We used two metrics in our experiments. We looked at planner performance on the problems by evaluating the time it took to plan paths for each problem. We also measured the ability of a real robot to execute the plans returned by our planner. An execution was considered a "success" if the object (plate or bowl) was placed in the goal position without disturbing the rest of the environment (i.e. knocking anything off the table). The robot executed these plans "open-loop" in that it sensed the object's position once before planning and never again during execution.

The planner was able to find solutions for all four problems. Running times and execution success fractions are given in Table 1. In this table we also give results for a problem identical to Plate3 except that the goal was in the center of the environment rather than on a table. This problem is included to emphasize how difficult it is to plan to place a plate. For comparison's sake, we also include planning time for a problem in which the goal is in the center of the environment and there are no obstacles placed on the table around the plate. Videos of the trajectories executed by the PR2 in each of the domains are on our website[1].

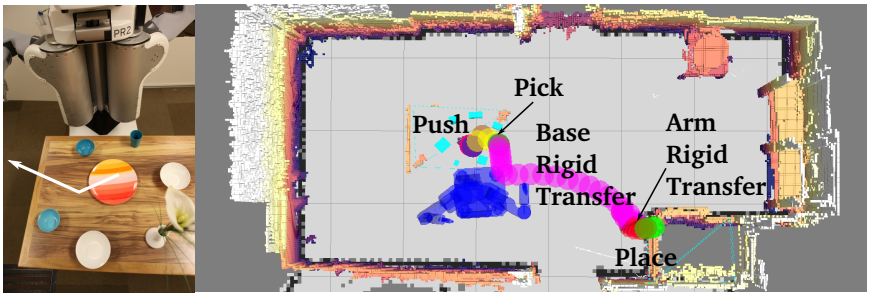
[1] `http://people.csail.mit.edu/jbarry/pr2/darrt`



(a) Plate1



(b) Plate2



(c) Plate3

Fig. 5 The plate domain problems. The 3D map of the obstacles present in every domain (walls and tables) is shown as a colored grid while obstacles that we added and removed from the environment (bowls, cups, etc) are shown as blue boxes. For each problem, we show the starting state of the robot and obstacles, possible trajectories for pushing the plate to the edge of the table (white arrows), and the planned trajectory for the plate color-coded by primitive. Videos are on our website[1].

6 Performance Analysis

The running times given in Table 1 are for an unoptimized version of the planner. Optimizing the planner will significantly decrease these times. However, we include them because they give a good metric of the relative difficulty of different problems.

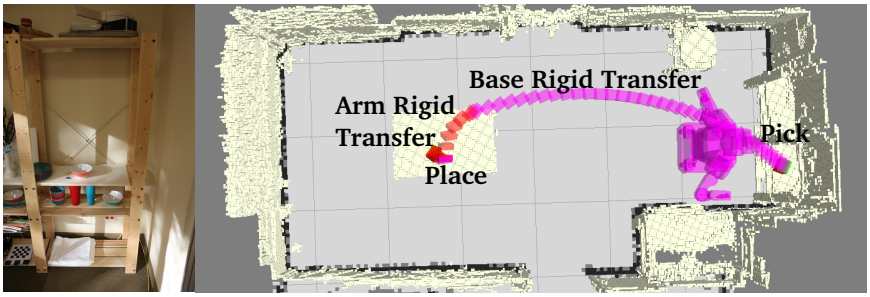


Fig. 6 The Bowl domain. Videos are on our website[1].

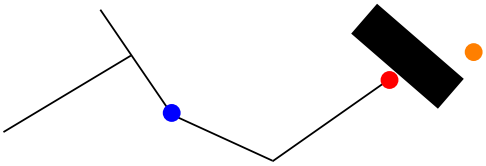
6.1 Problem Difficulty

It is clear from the running times in Table 1 that some problems are easier for the planner than others. In the Plate1 problem (Figure 5(a)), for instance, the plate can be moved to the edge of the table in a straight line parallel to the robot's torso. This is an easy manipulation for the robot and requires only three primitive instances (Arm-Transit, Straight-Line-Arm-Transit, and Push), making it relatively easy to find a plan in this domain. Similarly, the Bowl domain (Figure 6) is an easy domain with no pushing at all. We included this domain to show that the algorithm can work easily with other types of objects.

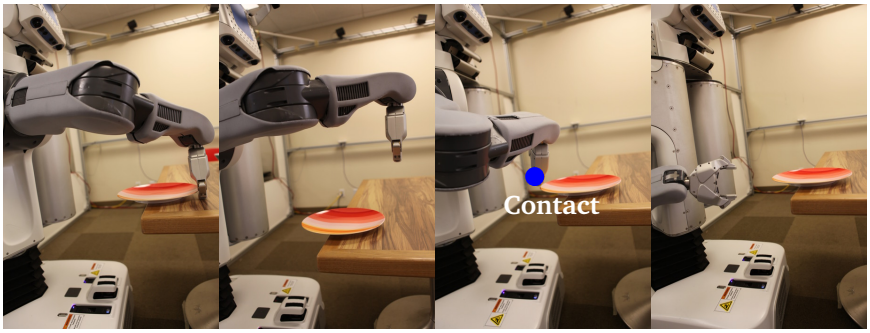
The Plate2 and Plate3 problems, however, require longer plans. In the Plate2 problem (Figure 5(b)), it is possible to move the plate to the edge of the table by pushing the plate back towards the robot using a single push or to the left edge of the table using a minimum of two pushes to move the plate around the small bowl. However, it is not possible to find an arm trajectory that can push the plate towards the robot without first moving the robot's base. In the Plate3 problem (Figure 5(c)) we removed all possible straight line paths to the edge of the table, forcing the planner to plan at least two pushes. This problem shows that the algorithm can find plans requiring multiple instances of a non-prehensile primitive.

6.2 Planning Time Analysis

We also analyzed our experiments to find the bottlenecks in planning. As with most sampling-based algorithms, the majority of the planning time is spent finding paths around obstacles. This is a problem common to almost all RRT implementations because the algorithm is greedy in its choice for the nearest state in the tree. For example, as shown in Figure 7(a), we usually quickly grow an RRT all the way towards an obstacle. Subsequent samples beyond the obstacle (shown in orange in the figure) find the point near the obstacle (shown in red in the figure) as the "nearest" point in the tree, but this point cannot be extended towards the sample. It takes a



(a)



(b)

Fig. 7 For most RRT based algorithms the majority of the time is spent finding paths around obstacles. (a) An example in the two dimensional case with a single obstacle (black rectangle). The initial state is shown in blue, a sample in orange, and the nearest point to the sample in red. (b) A similar example in our domain. When the plate is at the edge of the table the robot can grasp it. However, in trying to move from the pushing configuration (left) to the approach to the grasp (right), the gripper usually contacts the plate.

large number of samples to find a path around the obstacle. In our experiments, this problem was most evident in two situations: transitioning from pushing to grasping and placing on the table. We describe these scenarios in detail below.

When the plate is at the edge of the table, the robot can grasp it. However, in moving the plate to the edge of the table, the robot must have used the push primitive, which puts its gripper on the far side of the plate from the table edge, as shown in Figure 7(b). During the transition to the grasp, the robot retreats upwards from the push and then moves in a straight line in joint space to the approach to the grasp. In many cases, there is a collision between the plate and the robot's gripper along this line. Subsequent samples for the plate will almost all be nearest to this state because both the plate and the robot are close to a state in which the robot can approach a grasp for the plate. However, the state in the tree cannot be extended directly towards the approach to the grasp because of the gripper-plate collision. In order to move around the plate, the gripper must first move off the direct line to the approach to the grasp, around the plate, and then to the approach to grasp configuration. This requires a large number of samples in the robot's subspace.

The table on which we placed the plate is approximately one meter tall, which makes it too high for the robot to manipulate upon easily, especially as the gripper must be angled when placing flat plates. In addition, this table is about 25 centime-

ters higher than the table from which we picked the plate, which means that if the robot does a base transfer directly from the pick to the place, the plate will hit the side of the table as the robot moves it towards the place position. This creates a state in which the plate is near the place position but cannot be moved directly there.

To understand the relative difficulty of placing versus transitioning from pushing to grasping, we also considered a problem in which the robot had to push the plate off of the table and transfer it to a specific position in the environment but not place it. One might expect pushing to be harder than placing, because the relative configurations of the robot and object are more constrained. However, because of the particulars of our test environment, the placements are more constrained. In particular, the central table where the pickup happens is one under which the robot can move its base whereas the table for the place has a solid base. Without placing, the planning time was not much over one minute.

7 Future Work: Accounting for Uncertainty

The planner does not take into account uncertainty in the world. Moreover, the execution is open-loop in that we sense the plate's position once at the beginning of the planning and never again throughout execution. This gives us two main sources of uncertainty: uncertainty in the initial detection and uncertainty in the robot's motion. Slightly incorrect gripper positions originating from one of these tended to compound so that domains requiring more pushes (Plate3) or domains requiring long pushes (Plate1) tended to have execution failures[2] more often.

We are considering several approaches to dealing with uncertainty. For uncertainty created by the robot's motion, we can use a re-planning strategy [10], detecting when we have deviated from our planned path and updating the path accordingly. For perception error in the initial detection, we can use closed-loop control, checking the pressure sensor on the robot's fingertips for contact with the plate and adjusting. With more sensor input during the execution, we can also use strategies that plan in belief space [15] to try to bias the planner towards actions that also gain information about the environment.

Another method for minimizing uncertainty in execution is to choose primitives to reduce the uncertainty as much as possible. For example, we use two point pushing rather than single-point pushing. By using two point pushing, placing one side of the gripper on each of the center of friction, the robot is able to control the plate much more effectively [13]. We also implemented base and arm movement separately because the uncertainty models for the PR2 arms and base are very different, making it difficult to accurately execute synchronous trajectories. In general, separating primitives with different uncertainty models will also allow us to re-plan after executing primitives we know are likely to result in an uncertain state.

[2] `http://people.csail.mit.edu/jbarry/pr2/darrt#failure_modes`

We have shown that the DARRT algorithm can solve problems requiring the use of many types of manipulation in a sixteen degree-of-freedom space. However, we also found that the trajectories returned by the planner could be successfully executed only about sixty percent of the time. We have given both an analysis of the planner and directions for future work in reducing and planning for uncertainty.

References

1. van den Berg, J., Stilman, M., Kuffner, J., Lin, M., Manocha, D.: Path Planning among Movable Obstacles: A Probabilistically Complete Approach. In: WAFR (2008)
2. Brost, R.C.: Automatic Grasp Planning in the Presence of Uncertainty. IJRR **7**(1) (1988)
3. Cosgun, A., Hermans, T., Emeli, V., Stilman, M.: Push Planning for Object Placement on Cluttered Table Surfaces. In: IROS (2011)
4. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine (2012). URL `http://ompl.kavrakilab.org`. To appear.
5. Dogar, M., Srinivasa, S.: Push-Grasping with Dexterous Hands: Mechanics and a Method. In: IROS (2010)
6. Dogar, M.R., Srinivasa, S.S.: A Framework for Push-Grasping in Clutter. In: RSS (2011)
7. Hauser, K.: Motion Planning for Legged and Humanoid Robots. Ph.D. thesis, Stanford University (2008)
8. Hauser, K., Ng-Throw-Hing, V.: Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task. IJRR **30**(6) (2011)
9. Huang, W., Mason, M.T.: Experiments in Impulsive Manipulation. In: ICRA, vol. 2 (1998)
10. Kaelbling, L.P., Lozano-Pérez, T.: Pre-Image Backchaining in Belief Space for Mobile Manipulation. In: ISRR (2011)
11. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
12. Lozano-Pérez, T., Jones, J.L., Mazer, E., O'Donnell, P.A.: Handey: A Robot Task Planner. MIT Press, Cambridge, MA (1992)
13. Mason, M.T.: Mechanics of Robotic Manipulation. MIT Press, Cambridge, MA (2001)
14. Okada, K., Haneda, A., Nakai, H., Inaba, M., Inoue, H.: Environment Manipulation Planner for Humanoid Robots Using Task Graph That Generates Action Sequence. In: IEEE/RSJ IROS (2004)
15. Platt, R., Tedrake, R., Kaelbling, L., Lozano-Pérez, T.: Belief Space Planning Assuming Maximum Likelihood Observations. In: RSS (2010)
16. Siméon, T., Laumond, J.P., Cortés, J., Sahbani, A.: Manipulation Planning with Probabilistic Roadmaps. IJRR **23**(7-8) (2004)
17. Stilman, M., Kuffner, J.: Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments. In: HUMANOIDS (2004)
18. Stilman, M., Kuffner, J.: Planning Among Movable Obstacles with Artificial Constraints. IJRR **27**(11-12) (2008)
19. Stilman, M., Schamburek, J.U., Kuffner, J., Asfour, T.: Manipulation Planning Among Movable Obstactions. In: ICRA (2007)