Robotics: Science and Systems 2014
Berkeley, CA, USA, July 12-16, 2014

Planning Single-arm Manipulations with N-Arm Robots

Benjamin Cohen
bcohen@seas.upenn.edu
University of Pennsylvania

Mike Phillips
mlphilli@andrew.cmu.edu
Carnegie Mellon University

Maxim Likhachev
maxim@cs.cmu.edu
Carnegie Mellon University

***Abstract*—Many robotic systems are comprised of two or more arms. Such systems range from dual-arm household manipulators to factory floors populated with a multitude of industrial robotic arms. While the use of multiple arms increases the productivity of the system and extends dramatically its workspace, it also introduces a number of challenges. One such challenge is planning the motion of the arm(s) required to relocate an object from one location to another. This problem is challenging because it requires reasoning over which arms and in which order should manipulate the object, finding a sequence of valid handoff locations between the consecutive arms and finally choosing the grasps that allow for successful handoffs. In this paper, we show how to exploit the characteristics of this problem in order to construct a planner that can solve it effectively. We analyze our approach experimentally on a number of simulated examples ranging from a 2-arm system operating at a table to a 3-arm system working at a bar and to a 4-arm system in a factory setting.**

I. Introduction

Robotic arms often share a common workspace. Examples range from dual-arm manipulation platforms designed for household scenarios to factory floors equipped with dozens of industrial manipulators performing automated assembly (Figure 1). Yet, despite sharing heavily overlapping workspaces, these arms are frequently treated as independent entities, each arm performs its task independently and collaboration between the arms is minimal. On the other hand, effective cooperation between these arms can dramatically increase their productivity and overall cost-effectiveness. For instance, they can share common tools by passing them to each other when necessary, they can lift heavier object using multiple arm support and they can pick up assembly parts from remote locations and pass them to the arms requiring them for the assembly. For many of these tasks, one of the key requirements is the ability to plan arm motions that relocate an object from one location to another in the workspace spanned by the N-arm robotic system.

This planning problem is challenging for several reasons. First, it requires reasoning over which arms and in which order should manipulate the object. In the simplest scenarios a single arm may be able to relocate the object. In other scenarios, all N arms may potentially be involved if the object is relocated from one end of the workspace to the opposite end. Furthermore, sometimes, the same arm may have to regrasp the object several times with the help of another arm in order to grasp the object in a way that allows the arm to place the object at the goal with the desired orientation. Second, the planning problem also involves computing valid locations for each of the handoffs between two consecutive arms. Figuring out these locations is non-trivial because the environment can be cluttered and the object itself can be a large object. Finally, the planner also needs to consider different possible grasps and plan these grasps in a way that allows for the successful sequence of handoffs.

Fig. 1: Shown above are multiple arms operating on the same car in a common factory scenario. Despite sharing heavily overlapping workspaces, industrial arms are used quite often as purely independent workers.

In this paper, we show how to exploit the characteristics of this problem in order to construct a planner that addresses these challenges within a single search. In particular, we introduce a compact representation of the planning problem, develop a heuristic graph search algorithm that exploits the fact that some of the computationally challenging operations can be postponed and show how an effective heuristic function can be derived. We analyze our approach experimentally on a number of simulated examples ranging from a 2-arm system operating at a table to a 3-arm system working at a bar and to a 4-arm system in a factory setting.

This research was partially sponsored by ARL, under the Robotics CTA program grant W911NF-10-2-0016. We are also thankful to Google for their partial support of this work.

II. Related Work

There has been a lots of work on motion planning for manipulators. Sampling-based motion planners [8, 10, 2] are among the most popular due to their efficiency at solving high dimensional problems. Other more recent approaches for manipulation involve optimization techniques [13] and discrete graph searches [3]. Another work [5] presents an efficient method for planning paths for a set of arms to reach their goals. These types of planners make up one part of the problem we are solving. In fact, while our method uses one such motion planner, any of these could be used for that part.

There has also been some work on computing handoffs between objects where the required handoffs are known a priori. In this work [16], the authors use a sampling based method to solve the problem of moving an object from one place to another using multiple arms. The approach allows for scenarios where a fast IK solver is not available whereas our method requires IK for our representation. In [14] the authors solve the problem of finding good grasps to use when a handoff between two arms is known to be needed. Our approach solves what this algorithm assumes, determining what handoffs are needed. Some work has been done to use optimization techniques to find handoffs between two arms [1]. While all of these approaches address the problem of moving an object between hands, none of them solve the discrete problem of finding the sequence of arms required to move the object to its goal. Our approach solves this component of the problem as well.

This early work [9], is one of the most similar to ours. The authors have an approach to moving an object with several manipulators. The problem is solved by decomposing the discrete and continuous problems. First the discrete problem is solved by finding which arms will move the object and in what order. Then each of these steps is solved. The approach assumes that if the high-level plan calls for an arm to move to the object for grasping, the plan will succeed. This is not always possible, and by combining these two levels of planning our method overcomes this assumption.

The algorithm presented in [7] plans multi-step plans where each step involves a continuous plan using PRMs. While not presented in the context of our domain (the paper's experiments are on locomotion) it is applicable. We have adapted this to our domain and compared our planner against it in the experimental results section.

III. Problem Definition

The problem is to move an object from a start pose to a goal pose in SE(3) without colliding with obstacles in the environment. The object can only be moved by an arm that is currently grasping it. We are given a set of manipulators in the environment that can be used to move the object. Note that the set can contain different manipulators each with a different number of joints. We are also given the list of valid grasps for the object. The planner has to find a joint space plan for all the arms that results in moving the object from its start to goal pose where neither the object nor arms collide. This plan can include "handoffs" which allows the object to be transferred from one arm to another. Additionally, it is desirable to find a path that is reasonably short and smooth.

IV. Algorithm

The full dimensionality of this problem is immense as it includes all the degrees of freedom for each arm as well as the 6 DoF pose of the object. When planning for large numbers of arms as we might find in a factory this representation simply doesn't scale. One of key insights to our approach is that for the vast majority of scenarios only one arm is relevant at any given point in the movement of the object, except during a handoff where the configurations of two arms matter. Our novel representation plans for the 6D pose of the object floating through space while ensuring there is a "support arm" at each pose.

Our problem has both discrete and continuous components. The discrete component is finding the sequence of arms and grasps needed to support the object throughout its motion. The continuous problem comes from the manipulators and object moving through a continuous configuration space. Our algorithm uses a search-based planner (a variant of A*) which plans on a graph. Such planners excel at discrete problems [6] as graphs are inherently discrete and can use informative heuristics to focus the search toward the solution. In our problem, we use the heuristic to estimate the solution to the discrete problem of finding the sequence of arms used to transport the object. Search based planners have also been shown to perform comparably to their sampling-based counterparts in continuous planning problems such as manipulation [3]. Graph search approaches also makes it easy to handle path constraints (e.g. holding a pitcher upright), as well as goal sets or regions (e.g. place the object anywhere on the table). A search-based planner is a natural choice for this problem as it seamlessly combines the discrete and continuous parts of the problem. In our problem we will be planning for the object with a heuristic that guides the path of the object through the appropriate sequence of handoffs. Finally, in our planner, many of the possible actions will be expensive to evaluate. The main planner can actually call a single arm motion planner to determine the possibility and cost of a handoff motion. To reduce the overall planning time, a lazy variant of the popular weighted A* algorithm is presented in order to postpone expensive evaluations until they are absolutely necessary.

A. Notations and Assumptions

In order to reduce the dimensionality to something manageable and scalable, only the arm currently supporting (grasping) the object is used in our representation. To plan safely in the presence of the other, non-supporting arms, we assume that when an arm is not in use it is in a known "safe configuration". Note that a set of safe configurations for each arm can be used as well.

We will be using a substantial amount of graph search notation:

- $G(V, E)$ is a graph where V is the set of vertices (or states) in the graph and E is the set of edges connecting pairs of vertices in V.
- s_{start} is the start state.
- s_{goal} is the goal state.
- $c(u, v)$ is the cost of the edge from vertex u to vertex v.
- $g(s)$ is the cost of the cheapest path from the s_{start} to s found by the algorithm so far (we will sometimes write g-value).
- $g^*(s)$ is the optimal (minimum) cost from s_{start} to s
- $h(s)$ is a consistent heuristic. It provides an underestimate of the distance to the goal and satisfies the triangle inequality.

B. Representation

We represent the planning problem with a graph. Since we are assuming that all arms not currently supporting the object are in their safe configurations, a state only needs to contain information about the pose of the object and the arm supporting it. There are 6 dimensions for the position and orientation of the object. There is also one variable representing the arm that is supporting the object and one for the grasp it is using. These two dimensions are just indices since we are given a finite lists of arms and grasps. Finally, while this is enough information to know where the gripper of the supporting manipulator is, for many manipulators this doesn't map to a single set of joint angles. There are often many solutions for a particular end effector pose. These manipulators can have this redundancy captured by a set of "free angles", denoted below as $\phi_0, \ldots, \phi_m$. For example, in our experiments we use PR2 arms which have 7 degrees of freedom. An end effector pose can be mapped to a set of joint angles given one free angle (in this case the upper arm roll joint). Formally a state vector is defined as:

$$[x_{obj}, y_{obj}, z_{obj}, roll_{obj}, pitch_{obj}, yaw_{obj}, arm_{id}, grasp_{id}, \phi_0, \ldots, \phi_m]$$

Edges in a graph connect pairs of states and represent the motions of one or more arms in the transition. We call these edges "motion primitives". Motion primitives are actions that can be applied at any state in the graph to generate neighboring states. Most of our motion primitives involve moving the object while using the same support arm. There are motions for modifying the object position and orientation state variables by small amounts. There are also motion primitives for changing the support arm free angles. All of these motions are computed by running inverse kinematics with the new object pose or free angle. The motions are then collision checked by interpolating between the two states. Our representation assumes a fast inverse kinematics method for all manipulators considered by the planner. For most arms, a generic freely available inverse kinematics library, such as IKFast [4] is sufficient. Note, that the "free angle" variables do not have to correspond to specific joint angles, but rather represent the degree of freedom of the redundancy. While our experiments were performed on 7 DoF arms, we believe that it can be used with higher degree of freedom arms by adding additional dimensions.

The last kind of motion primitive is to switch supports. To do this a newly selected arm must move from its safe configuration to the newly selected grasp and then the previous support arm has to move back to its safe configuration. These motions can be quite complicated so we call an arm planner for a short period of time to check for feasibility. The planner we chose to use is also a search-based motion planner [3]. The planner is called both to get the new support arm to the object, but also to return the old support arm back to its safe configuration. Evaluating this edge is very expensive since another planner needs to be called twice. In the next section we provide a solution to this expensive evaluation time.

C. Search

Our algorithm is search-based planner rooted in A*. All of these planners run on graphs which are discrete in nature. When planning for a manipulator which exists in a continuous configuration space, the space is discretized onto a grid. The objective of A* is to find an optimal path (sequence of edges) in the graph that connects a start state s_{start} to a goal state s_{goal}. A* does this by repeatedly expanding the state in $OPEN$ with the smallest f-value, defined as $f(s) = g(s) + h(s)$. $OPEN$ is the list of states that have been discovered by the search but haven't been expanded yet. An expansion means to generate the state's successors and put them in $OPEN$. A state only needs to be expanded once.

Our algorithm is a variant of Weighted A* [12]. Weighted A* inflates the heuristic component of the f-value by $\varepsilon > 1$ causing the search to be more goal directed and find solutions significantly faster than A*. Weighted A* is not optimal but is guaranteed to return a solution no worse than ε times the optimal solution cost even if each state is expanded at most once [11].

When planning in high-dimensional problems the graph is too large to compute up front and is instead generated as the search progresses. One of the characteristics of our problem is that some edges in our graph are far more expensive to validate than others. In particular, determining if the object can switch support arms requires two calls to an arm planner. Additionally, most states have the option to transition to more than one arm for several different grasps. Expanding a state in the usual fashion, i.e. generating all successors and putting them in $OPEN$ would result in each expansion taking a huge amount of time. Under this traditional approach most handoffs will get computed but since they don't actually get the object closer to the goal (compared to a motion with the same arm which moves toward the goal) the planner will never expand them. This is a massive waste of computation time.

We propose a lazy version of Weighted A* which fully evaluates edges only when the planner intends to use them. This is done by giving the edge an optimistic value for its cost and putting it in $OPEN$ We say this edge does not have its *true cost*. Then, only when the state is selected for expansion, do we evaluate the edge (in our domain this involves calling arm planners to switch support arms). After the edge it evaluated we may find that it is invalid, in which

case we throw it out. If it is valid then we now know the edge's true cost (and it's probably more expensive than our optimistic guess) so we put it back into $OPEN$ and when it comes out the second time, it will actually be expanded and will generate successors of its own.

LazyWeightedA* shows our lazy version of weighted A*. One of the most important differences is that a state can actually have several copies of itself in $OPEN$ with different parent states. This isn't required in normal weighted A* since only the cheapest path to a state needs to be kept. However, since many of the edges haven't been evaluated, we don't know the true cost of the states and it's possible the instance of the state with the cheapest cost may actually use an edge which turns out to be invalid after it's evaluated. Therefore, we need to maintain duplicate states in $OPEN$.

Lines 1-2 of LazyWeightedA* are typical of weighted A*. We iteratively remove the cheapest state in $OPEN$ until the goal is the cheapest. Lines 3-4 are new and show that if a state has already been expanded it doesn't need to be expanded again. $CLOSED$ contains the states that have already been expanded. While $OPEN$ allows for duplicates, $CLOSED$ does not. Typically, these lines are not needed for weighted A*, but in the lazy variant, multiple copies of a state could be in $OPEN$ so the ones that come out after s has been $CLOSED$ have larger costs and can be ignored. Typically, in weighted A*, when a state is removed from $OPEN$ it is expanded. We can see in the lazy version, this depends on if we know its true cost or not (line 5).

If the state's cost is true, we do an expansion by first marking the state as closed (line 6) and getting a copy of each neighboring state (line 7). The set S contains copies of each state. The function getSuccessors also marks some states as having trueCost(s') set to true and other set to false. This is domain specific and depends on which edges you want to evaluate lazily. Any state that does not have its true cost (being evaluated lazily) must have a non-overestimating cost for the edge $c(s, s')$. This is needed to guarantee bounded sub-optimality of the solution cost. On line 9 we see that if a state has been $CLOSED$ (already expanded) we can ignore this successor. On line 10 we set the new state's parent to be s. On line 11, we compute a g-value for this version of the state s' (coming from this parent s). On line 12 we see if this version of s' is worth keeping for consideration ($conf(s)$ represents the actual configuration that corresponds to s). If we already have a copy of s' that has a better g-value (and the g-value is a true cost) then we don't need this new version. On lines 13-14, we compute its f-value and put it into $OPEN$.

Lines 19-25 show the case where a state s comes out of the $OPEN$ list but doesn't have its true cost yet. This means the edge from the parent of s to s needs to be validated. The getTrueCost function is called on line 19 to do this. If the cost is finite, the edge exists and we mark the state as having a true cost and update its g-value. Like the previous computation of the f-value and insertion into $OPEN$ (lines 24-25), we only do it if there isn't a better version of the state (line 23).

This lazy variant of Weighted A* is broadly applicable. It is both complete and has bounded suboptimality (solutions are guaranteed to cost no more than ε times the cost of an optimal solution) with respect to graph used to represent the problem. However, it only makes sense for domains where evaluating edges are costly and will offset the additional insert and remove operations that we perform on $OPEN$. Additionally, the memory footprint of the lazy variant is generally worse than Weighted A* due to maintaining duplicate states.

LazyWeightedA*()

```
1:  while f(s_goal) > min_{s∈OPEN}(f(s)) do
2:      remove s with the smallest f(s) from OPEN
3:      if s ∈ CLOSED then
4:          CONTINUE
5:      else if trueCost(s) then
6:          insert s into CLOSED
7:          S = getSuccessors(s)
8:          for all s' ∈ S do
9:              if s' ∉ CLOSED then
10:                 parent(s') = s
11:                 g(s') = g(parent(s')) + c(parent(s'), s')
12:                 if ∄s'' ∈ OPEN s.t. conf(s'') = conf(s') ∧
                    trueCost(s'') ∧ g(s'') ≤ g(s') then
13:                     f(s') = g(s') + εh(s')
14:                     insert s' into OPEN
15:                 end if
16:             end if
17:         end for
18:     else
19:         c' = getTrueCost(parent(s), s)
20:         if c' < ∞ then
21:             trueCost(s) = true
22:             g(s) = g(parent(s)) + c'
23:             if ∄s'' ∈ OPEN s.t. conf(s'') = conf(s)∧trueCost(s'')∧
                g(s'') ≤ g(s) then
24:                 f(s) = g(s) + εh(s)
25:                 insert s into OPEN
26:             end if
27:         end if
28:     end if
29: end while
```

D. Heuristic

Our heuristic has two components: guiding the object toward the goal and encouraging the most promising handoffs.

In order to guide the search toward the goal, we run a reverse Dijkstra search for the 3D location of the object backward from the goal to all (x,y,z) cells in the environment. We call this a "Point Search". This approximates the distance from all positions in the world to the goal location. Additionally, this search is computed with the obstacles, so the heuristic will guide the object around obstacles (note that since we are not searching over orientation, we have to use a sphere that fits inside the object, so it only guides a small part of the object around obstacles). In order to capture some of the orientation information of the goal pose, we actually run several of these Point Searches for different points on the object. By guiding a set of points on the object toward their goal positions, we help the search achieve the goal orientation. It is also much more efficient to compute several 3D searches than one 6D search. Formally, we will define the ith Point Search as $PS(s, i)$, where s is the state we are getting the heuristic for. PS uses

the position and orientation of the object in state s in order to locate the position of the ith point in the world and then report the distance of that point to its goal location.

The second component of the heuristic is a handoff penalty. We approximate the number of remaining handoffs needed to get the object at the goal pose. This approximation is formulated as a dynamic programming problem and is shown in numHandoffs. The function finds the minimum number of handoffs required to get to the goal assuming that the object is being held by arm with $grasp$. Lines 3-4 shows the base case of already being able to put the object at the goal. This requires 0 handoffs. Lines 6-12 show the general case. We will handoff to the neighboring arm with a valid grasp (given the one we're using) that requires the minimum number of remaining handoffs to reach the goal. We then add 1 to that for the handoff we need to get to that arm. The availableGrasps function uses a look up table to see what grasps are possible given the one being used (a grasp can invalidate many others due to collisions).

Finally, there is a base case for a maximum depth (lines 1-2). It's possible that there may be a huge number of arms and grasps so searching over all of them may be expensive to do. Also, it often doesn't make sense to worry about which grasp is being used more than 2 or 3 arms out from the goal. An obvious exception is an object with only 2 grasps being passed down a linear chain of manipulators. Since each handoff alternates grasps, in order to have the proper grasp at the last arm to put the object at the goal, it becomes critical that the first arm chooses its grasp carefully. We found that the maximum depth is mostly dependent on the object being manipulated and how many grasps it has. By returning 0 when the maximum depth is reached, we are guaranteed not to overestimate the number of remaining handoffs to the goal.

numHandoffs(arm,grasp,depth)

```
1:  if depth = MAX_DEPTH then
2:      return 0
3:  else if canReachGoal(arm,grasp) then
4:      return 0
5:  else
6:      minVal = ∞
7:      for all a ∈ neighborArms(arm) do
8:          for all g ∈ availableGrasps(grasp) do
9:              minVal = min(minVal, numHandoffs(a,g,depth+1)+1)
10:         end for
11:     end for
12:     return minVal
13: end if
```

Incorporating the number of remaining handoffs into the heuristic function provides critical guidance to the search. It discourages using unneeded handoffs because the heuristic does not decrease if the action does not reduce the remaining handoffs. This is useful since evaluating those edges is expensive. It also encourages needed handoffs because the heuristic will drop on handoff edges that are closer to an (arm,grasp) pair that goes to the goal. This computation will also help the planner to choose the right grasps.

This handoff count is scaled by a conservative cost for any handoff in our system. Since we know any handoff involves one arm going from its safe configuration to a grasp and another arm going from a grasp to it safe configuration, an underestimate of all handoffs would be

$$HP = \max\left(0, \min_{a0,a1 \in Arms}(d(a0,a1)) - \max_{g0,g1 \in Grasps}(d(g0,g1))\right)$$

We call HP our handoff penalty. $Arms$ here is the safe configuration of each arm and $Grasps$ is the set of grasps on the object. The distance function d is just a simple euclidean distance for the end effector (it just needs to be conservative).

We combine all the heuristic components into the following:

$$h(s) = HP \cdot \text{numHandoffs}(arm_{id}(\text{s}), grasp_{id}(\text{s}), 0) + \frac{1}{|PS|}\sum_{i=1}^{|PS|} PS(s,i)$$

Since our heuristic needs to be admissible (an underestimate), we ensure that the handoff term underestimates the handoffs and the point search term underestimates object motion (note that we don't have any motion primitives that cause a handoff and move the object, so there is no overlap between these two terms). For the Point Searches we add the distance remaining for each point to its goal location, but then divide by the number of Point Searches (giving an average distance for the object to the goal as represented by these points). This doesn't overestimate because an average is guaranteed to be no bigger than its largest contributing value. Finally, the handoff term underestimates because we chose HP to underestimate any possible handoff, and the numHandoffs function is a conservative guess for the number of remaining handoffs.

V. Experimental Results

To measure the performance of the planner, we generated a set of realistic scenes in which we strategically placed arms such that almost all areas of interest can be reached by at least one. The manipulators used are the 7 DoF arms of the PR2 robot. In this section, we present a battery of tests demonstrating the capabilities of the approach, including the efficient planning times. A comparision to alternatives is also provided.

A. Implementation Details

Given that the highest number of degrees of freedom of any arm in the set available to the planner is seven, the position of only a single redundant joint has to be recorded in the state representation, resulting in a total of nine dimensions. The statespace is discretized as follows. The translational dimensions are discretized with 2cm resolution and the rotational dimensions, including the joint position of the free angle as well as the object pose, have 2° resolution. Our experiments included between 3 and 4 arms and the set of predefined grasps ranged from 12 to 14.

The set of motion primitives employed by the planner includes twelve actions, each of which translates or rotates the

object along one dimension. The set also includes two primitives that rotate the free angle in each direction. Additionally, we use *adaptive motion primitives* [3] that use two analytical solvers to snap to the goal pose with a single motion when the search is within range. These motions are capable of achieving any arbitrary goal pose despite the resolution of the graph. The set of motions also includes *switch support actions*, or the act of performing a handoff. The number of switch support actions at a given state is determined by the number of grasps defined in the predefined set multiplied by the number of neighboring arms within arms reach of the current support arm. Thus, the branching factor of the graph and the speed with which it is explored is highly dependent on the number of neighboring arms and the size of the set of grasps.

Evaluating if a switch support action is valid is an expensive task. It requires that two paths are computed, one by the receiving arm, from the safe pose to the specified object grasp, followed by a path by the current support arm to return to its safe pose once the handoff is complete. Unfortunately though, the single arm path planner employed by our planner is incapable of planning to or from a state in collision. Thus, the validation of handoff action occurs as follows. Given the specified grasp, we compute a pregrasp pose that is offset from the object by a set amount that would remove the gripper from its invalid state. If an IK solution exists for both the grasp and the pregrasp pose and they are both determined to be collision free then a single arm planner is used to plan a path from the safe pose of the arm to the pregrasp and then an open loop motion from the pregrasp to the grasp would complete the first half of the handoff operation. A similar process is performed on the letting go part of the action. First, we compute an IK solution for the postgrasp pose, which in our case, is the same offset from the object as the pregrasp. If the solution exists and is valid, then a motion plan from the postgrasp pose to the safe pose is requested of the single arm planner. If a solution exists, then at this point, we are sure that the entire handoff action is considered valid and an edge can be added to the graph. Note that for the object to be picked up in the first place, a very similar action is used. Instead of planning for two arms, only one is required to switch between a support surface and a support arm. In our implementation, we use a search-based planner [3] for single-arm planning and we give it a maximum of 1 second to compute a path. The Lazy Weighted A* is complete and has bounded suboptimality in general. If we always allowed the single arm planner to run to termination (either find a solution or exhaust the graph) our overall approach would be resolution complete and bounded suboptimality with respect to the graph. However, since we restrict it to planning in 1 second, we forfeit these guarantees.

The overall cost the planner minimizes is the sum of the distances traveled by each gripper. The cost of handoff motions are initially estimated by summing the straight line distance between the current support gripper and its safe pose and the straight line distance between the soon to be support gripper at its safe pose, and the specified grasp on the object. This approximation is optimistic, so we can use it for lazy edges.

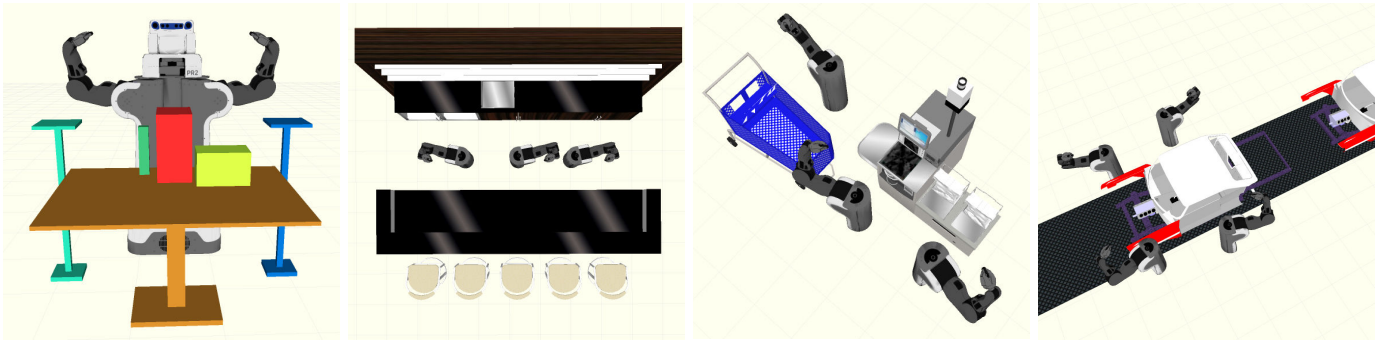
B. Experimental Setup

We use four different scenes in our experiments. Depending on the scene, between two and four arms are available to the planner to accomplish the goal. The planning problems in each scenario were created by hand to assure that each one met two requirements. The first is that at least a single handoff is required, meaning a single arm can not perform a valid motion to go from the start to the goal pose given kinematic constraints or obstacles in the environment. The second requirement is for the tests to be challenging, either because of obstacles in the environment, kinematic constraints or restrictive start and goal pose orientations. The start and goal pose orientations of the object play a large role in the complexity of the trial because the object's initial orientation is a large factor in determining the set of feasible grasps to pick it up with by the neighboring arms. Given that the goals are defined as 6 DoF poses, it is easy to imagine a scenario in which an additional 180° rotation of the object at the start pose, may require that one or more additional handoffs beyond what is already needed are required if the set of grasps contains only one grasp per side.

The scenarios used for testing are as follows (Figure 2):

1) **Tabletop (2 arms, 5 trials):** - The PR2 robot is standing in front of a table with large obstacles on it.
2) **Bar (3 arms, 10 trials):** Three arms are placed at fixed locations behind the bar such that the entire width of the bar surface in front of them can be reached. Unfortunately, due to the restrictive shoulder pan joint in the PR2 arm, the counter top behind them can not be reached. Note, that below the bar are shelves that the arms can reach into.
3) **Self-checkout (3 arms, 5 trials):** The arms are placed such that one of them can empty the cart, while another arm can swipe the object in front of the bar code scanner and the last arm can place the object in the bagging area.
4) **Car Factory (4 arms, 5 trials):** Inspired by Figure 1, this scenario includes a conveyor belt transporting cars down an assembly line. Two arms are placed on each side of the car. The pair of arms closer to the hood are placed closer to the chassis so they can both reach into the engine block and pass things to each other as well. The rear arms are set a bit further from the body of the vehicle, forcing them out of each other's reach given that the objects are of a reasonable size.

Two different objects were used in these experiments. The first object is a narrow rectangular tray with dimensions, 10cm x 40cm x 2cm. The set of grasps includes 12 grasps in which 5 are placed equidistant from each other along the length of each side. There is also one grasp at each end. Note, that since the tray is especially narrow, a handoff would be infeasible in which two grasps are used that are directly opposite each other along the length of the tray. The second object is a rod with dimensions, 2cm x 30cm x 2cm. We defined 14 grasps for the rod which includes one at each end. The tray is used in all the experiments except for the self-checkout scenario.



(a) Tabletop (b) Bar (c) Self-checkout (d) Car factory

Fig. 2: Shown above are the four scenarios used in our experimental results.

	success rate	mean time(s)	max time(s)	mean expands	mean handoffs
tabletop (2-arm)	100	10.25	36.52	226	1.33
bar (3-arm)	50	16.93	39.93	857	2
checkout (3-arm)	100	9.67	16.22	220	1.8
factory (4-arm)	100	36.09	78.41	2239	1.8

TABLE I: Performance of the planner in four different scenes.

In each trial, the initial pose of the object, the goal pose and the set of object grasps are given to the planner. The planner returns a sequence of trajectories, labeled with which arms they are for. Note that the initial state of all of the arms is for them to be in their safe configurations. The planner is given a maximum of 100 seconds to compute a solution. A translational tolerance for the goal is specified as a 3cm cube in which the center of the object is supposed to end up and a tolerance of 0.05 radians is allowed on the object's final roll, pitch and yaw. All experiments were performed on a computer with an Intel i7 CPU (2.8Ghz), 16GB of RAM, running Ubuntu 12.04.

After a solution is found, we use a simple deterministic shortcutter to safely shorten the paths if possible. To speed up the execution of each handoff procedure, one can collision check the return-to-safe-pose trajectory of the arm that is handing off the object against the subsequent portion of the newly appointed support arm's trajectory. They can be executed simultaneously if deemed valid to do so. In our experiments, we found that while it is dependent on the relative positioning of the arms, in a large portion of our trials the trajectories did not cross paths and it would allow for a nice reduction in execution time.

Results from the trials can be seen in Table I. Overall, the average planning time is 18.2 seconds and the planner was successful in planning in 20 of 25 total trials, for a total success rate of 80%. In particular, we found that in most of the trials the minimum number of handoffs required were performed but not in all of them. Additionally, we were pleased to see that the planner successfully determined that because of the initial object pose, a series of back and forth handoffs were needed between the arm used to put the object down and the arm that transferred the object to it, so that the final grasp was capable of achieving the goal pose.

C. Naive Approach

We compared the performance of our planner to a naive approach to planning when only two arms are available. We made the problem a bit simpler for the naive approach by choosing trials in which the start and goal poses do not reside in the same arm's workspace. This means that a handoff is always required and deciding which arms perform the pickup and the putdown is easy. In this approach, handoff configurations (object pose and grasp for each arm) are sampled until one is found that passes an exhaustive approval process as explained below. If it fails at any point along the way, the sample is dropped. The approach continues sampling possible handoff configurations until one is found or a timeout is reached.

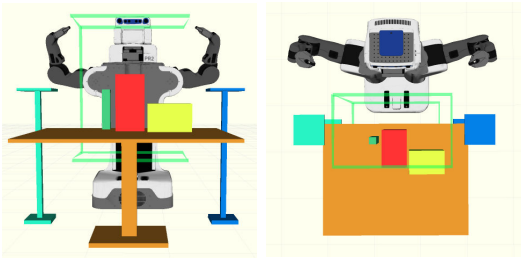


Fig. 3: The arms are shown in their safe poses in front of the tabletop scenario. The green line outlines the sampling region in which a handoff pose will be chosen by the naive approach.

In more details, this method tries to compute a valid handoff pose by randomly sampling a 6 DoF pose within the defined sampling region which is chosen manually based on the overlapping areas of the two workspaces (Figure 3). After a sample is generated, the object is checked for collisions with the environment. Then it randomly samples a $grasp_{id}$ for each arm. If an IK solution exists for each arm at its respective grasp and both are collision free, then we compute a pregrasp for the receiving arm and a postgrasp for the handing off arm in the same way that it is performed to evaluate a switch support action and check if IK solutions exist for both and are collision free. At this point, a pair of grasps that are valid at the handoff pose is known and we need to see if the same holds true at the pickup and putdown locations. IK solutions are computed

	naive success rate	naive mean time(s)	planner success rate	planner mean time(s)
tabletop	90%	18.39	100%	10.245

TABLE II: Performance comparison between the naive approach and our planner.

using those grasps and object poses and checked for validity. At this point, five paths are needed to connect the object pickup to the handoff and then to the object putdown. Note that this includes returning the pickup arm to its safe pose. If at any point in this process a step fails (i.e kinematically infeasible, in collision, planner fails to compute path), then the samples are discarded and new ones are generated.

In this approach, we use RRTConnect to plan all of the required paths. RRTConnect is one of the most common and fastest sampling based methods used today. We use the highly optimized implementation found in the OMPL [15].

Given that this method is limited to two arms, we are only able to run it on the tabletop scene. Since it relies on random sampling, we ran each trial 4 times and the entire set of trials is averaged together. The planner was only run once per trial. Results can be seen in Table II. Note that failures in the naive approach indicate that no solution was found within the 60 second timeout.

D. Incremental Multi-Modal PRM

We compared our planner to the Incremental Multi-Modal PRM (Incremental-MMPRM) [7]. This method is designed to plan multi-step plans. The phases are called modes and each constructs its own PRM. Where modes overlap, transition states are sampled to allow the PRMs to connect to one another in order to produce a multi-modal plan that passes through the different phases. For problems with multi-step structure, this is far more efficient than sampling in the full configuration space since some of the modes may have zero volume in the full configuration space and may be nearly impossible to sample.

We applying Incremental-MMPRM to our manipulation problem, creating modes for the various phases of the problem such as one arm moving while the object is at the start, one arm moving while holding the object, one arm holding the object while another moves freely, etc. While some examples of intersections between modes would be an arm grasping the object while it is at the start or two arms meeting in a handoff.

We gave 6 different problems to the planners (3 from the tabletop and 3 from a kitchen scenario). Since Incremental-MMPRM is randomized we averaged the runs over 10 trials on each problem. Table III shows the results of these experiments. The path length is computed as the sum of the distances traveled by each end effector. In general, our method has a higher success rate, similar planning times, and shorter path length (shortcutting is applied Incremental-MMPRM paths).

method	success rate	mean time(s)	mean path length (m)
Our planner	100%	5.26	5.61
Inc-MMPRM	76.7%	6.80	8.64

TABLE III: Comparison to Incremental-MMPRM

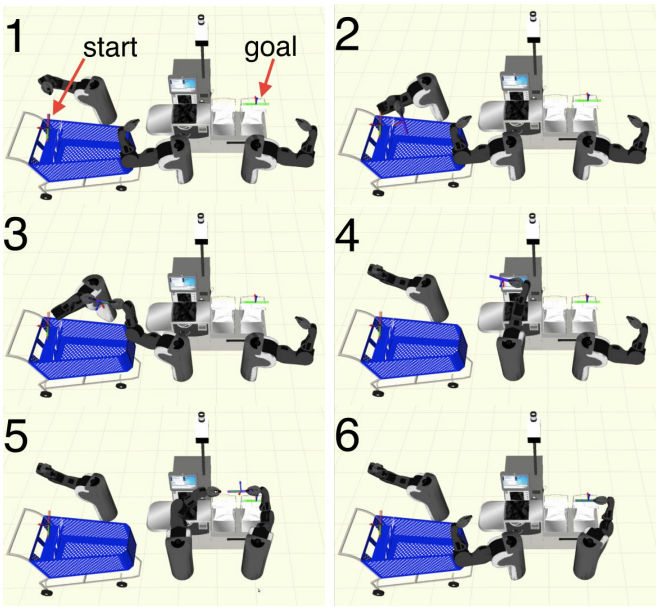


Fig. 4: Shown above, going from left to right, is a sequence of handoffs that successfully bring the rod from the cart to the bag.

VI. Conclusion

In this paper we address the problem of moving an object from one place to another given control of a multitude of manipulators that can pass the object amongst each other. This is challenging because it combines the discrete combinatorial planning problem of finding a sequence of manipulators and grasps to transport the object to its goal, with the continuous motion planning problem for each of those manipulators. Our algorithm seamlessly unites the two problems within a graph-based search which plans for the continuous motion of the object and the arms manipulating it, while being guided by a novel heuristic function that estimates a solution to the high level discrete problem. Our planner determines which arms and grasps will be used, where handoffs between arms will occur, as well as the motions of each of the arms. Additionally, our algorithm uses a lazy version of Weighted A* which postpones the evaluation of edge validity and cost to when the planner actually intends to explore them. This variant of the Weighted A* algorithm is more suited to planning problems where edges require significant time to evaluate, such as when another planner is used for that computation. We verified the effectiveness and scalability of our method through a variety of simulation experiments controlling up to 4 arms in practical scenarios (e.g. factory, grocery checkout, and bar). In the future we are interested in including the ability for the planner to place objects on support surfaces (e.g. a table or counter) in order to re-grasp them in a different way. This may allow the planner to accomplish even more complicated tasks.

References

[1] Benjamin Balaguer and Stefano Carpin. Bimanual regrasping from unimanual machine learning.

In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3264–3270. IEEE, 2012.

[2] Robert Bohlin and Lydia Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation, VOL.1*, 2007.

[3] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, 33(2):305–320, 2014.

[4] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008.

[5] Mokhtar Gharbi, Juan Cortés, and Thierry Siméon. Roadmap composition for multi-arm systems path planning. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 2471–2476. IEEE, 2009.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[7] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 29(7): 897–915, 2010.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] Yoshihito Koga and J-C Latombe. On multi-arm manipulation planning. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 945–952. IEEE, 1994.

[10] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

[11] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.

[12] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[13] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, 2009.

[14] Jean-Philippe Saut, Mokhtar Gharbi, Juan Cortés, Daniel Sidobre, and Thierry Siméon. Planning Pick-and-Place tasks with two-hand regrasping. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4528–4533. IEEE, 2010.

[15] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[16] Nikolaus Vahrenkamp, Dmitry Berenson, Tamim Asfour, James Kuffner, and Rudiger Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2009.