Search-based Planning for Dual-arm Manipulation with Upright Orientation Constraints

Benjamin Cohen Sachin Chitta Maxim Likhachev

Abstract— **Dual-arm manipulation is an increasingly important skill for robots operating in home, retail and industrial environments. Dual-arm manipulation is especially essential for tasks involving large objects which are harder to grasp and manipulate using a single arm. In this work, we address dual-arm manipulation of objects in indoor environments. We are particularly focused on tasks that involve an upright orientation constraint on the grasped object. Such constraints are often present in human environments, e.g. when manipulating a tray of food or a container with fluids. In this paper, we present a search-based approach that is capable of planning dual-arm motions, often within one second, in cluttered environments while adhering to the orientation constraints. Our approach systematically constructs a graph in task space and generates motions that are consistent across runs with similar start/goal configurations and are low-cost. These motions come with guarantees on completeness and bounds on the suboptimality with respect to the graph that encodes the planning problem. For many problems, the consistency of the generated motions is important as it helps make the actions of the robot more predictable for a human interacting with the robot.**

I. INTRODUCTION

Manipulation in human environments is a challenging and difficult task. Tasks that require manipulating large objects are particularly very difficult for single-armed manipulation systems. Examples of such tasks include the manipulation of large objects like laundry baskets, trashcans or boxes and manipulation of food or liquid containers, e.g. carrying a tray of food or glasses. Human execution of such tasks often requires the use of two arms. A general mobile manipulation robot attempting to execute such tasks will also require the use of two arms.

In this work, we focus on dual-arm manipulation, i.e. manipulation using two arms (Figure 1). Our focus is, in particular, on tasks where the robot is manipulating an object with an upright constraint throughout the motion. Such constraints often arise in human environments, particularly in transport of food or fluids, e.g when carrying a tray of food or a container of liquid. These tasks can often require manipulation in very cluttered environments, e.g. removing an open container of food from the back of a fridge while keeping it level.

A typical dual-arm manipulation task that we would like to execute is shown in Figure 1 where a PR2 robot is manipulating a tray with wine glasses on it. The robot is required to maintain the tray level throughout the execution of the task. The full execution of this task would be composed of multiple components including detection and tracking of the tray, determination of dual-arm grasps and motion planning and control for manipulation. In this work, we focus on the motion planning component of this task, i.e. the computation of collision free paths for moving the object from a start pose to a goal pose while maintaining the initial roll and pitch of the object.

Fig. 1: The PR2 robot manipulating a tray with two wine glasses on it.

Motion planning for dual-arm manipulation is inherently a constrained task. The act of holding an object with two hands naturally implies a constraint where the two end-effectors of the arms have to maintain a relative configuration with respect to each other. The rigidity of the grasp determines how much the end-effectors can move with respect to each other. We assume that the end-effectors are fairly constrained in moving relative to each other, i.e. the grasp being executed by the two arms is fairly rigid.

Our approach to this problem is based on the use of a heuristic search such as A* search [8] and its variants. The use of graph search-based planning techniques have been quite successful for low-dimensional planning such as for navigation. More recently, they have also been used for higher dimensional motion planning including single arm manipulation [5], [6] and some mobile manipulation tasks [4]. Search-based planning techniques typically have the advantages of good cost minimization and strong theoretical guarantees on completeness and optimality or bounds on the sub-optimality of the solution [17]. In addition, the graph

B. Cohen is with the Grasp Laboratory, University of Pennsylvania, Philadelphia, PA 19104 bcohen@seas.upenn.edu

S. Chitta is with Willow Garage Inc., Menlo Park, CA 94025 sachinc@willowgarage.com

M. Likhachev is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 maxim@cs.cmu.edu

representation can often incorporate complex constraints and wide range of cost functions. On the downside, the guarantees of search-based planning are with respect to the graph used to represent the planning problem rather than with respect to the underlying space. This implies that the guarantees are with respect to the used discretization of the state-space and action-space of the problem. Another important benefit of search-based planning is consistency in solutions: because of its deterministic nature and systematic discretization of the state- and action-spaces, the planned motions tend to be similar across runs in similar environments with similar start and goal configurations. This consistency and predictability of motions is often beneficial as it allows for humans to be more comfortable with the robot since they can learn to anticipate its actions in different environments.

In this paper we present an approach to dual-arm manipulation that relies on two key novel components to quickly generate feasible trajectories despite the complexity of the problem. The first component is the chosen representation of the state space that we use to efficiently reduce the dimensionality of the problem. The representation provided an additional bonus by reducing the discretization artifacts commonly found in the trajectories generated by our approach for single arm manipulation [6]. The second component is a novel heuristic that exploits the upright orientation constraint of the object to provide an even more informative heuristic to efficiently guide the search around clutter in the workspace. This paper will focus on those two concepts.

II. RELATED WORK

Manipulation of objects in human environments has gained a lot of interest in recent years ([19], [18], [7], [1], [11], [2], [22]). Interest in dual-arm manipulation is also gaining, particularly with the recent availability of dual-arm manipulation systems like the PR2 [3], Intel HERB Personal Robot [22], ARMAR [23] and Justin [2]. Dual-Arm manipulation has been used for tasks like cart-pushing [21], towel folding [16] and the manipulation of small kitchen objects [23].

Motion planning for dual-arm manipulation has mostly used randomized motion planners. In [23], randomized motion planners were used for planning re-grasping actions and dual-arm motion plans for a humanoid robot with two arms. Impressive results for combined grasping and motion planning were achieved by interleaving the efficient computation of inverse kinematics using a pre-computed reachability map with the motion planning itself. In [10], an assembly task was executed with two arms using combined task and motion planning. Online manipulation planning for objects moving on a conveyer belt was carried out in [14] for two 2-DOF arms. In [12], one of the first approaches to planning for dual-arm manipulation was presented using a randomized planner. Multi-arm manipulation planning, including the planning of transfer paths and grasp and ungrasp motions for manipulating an object using multiple arms was presented in [13]. The results of this approach were demonstrated on a simulated system manipulating a simple object using 3 arms.

The use of randomized planners for dual-arm manipulation tasks is popular because of the speed of the planners and ease of implementation. However, plans generated by such planners require frequent post-processing, e.g. using a shortcutting approach [9], before they can be executed on a robot. In addition, due to the random nature of the planners, the generated paths are also inconsistent across runs, i.e. paths planned for the same environment for similar start and goal positions are likely to be very different. Search-based planning addresses this issue providing consistency between runs and reasonable cost minimization.

III. THE PR2 ROBOT

The hardware platform that we used for experiments is the PR2 mobile manipulation robot (Fig. 1). The PR2 robot is a two-armed robot with an omni-directional base and a variety of sensors mounted on a sensor head. Each arm has 7 degrees of freedom and thus has a redundant degree of freedom. We exploit this redundancy by developing a custom inverse kinematics solution for the arm that is parameterized by one of the joint angles. The joint angle we choose is the *upper arm roll* joint shown in Fig. 3(b). The joint limits on the robot's joints are also taken into account by the kinematics solver. Thus, given the end-effector pose and a value for the free parameter, we can deterministically compute the corresponding inverse kinematics solution for this pose. In general, because of joint limits, we tend to find only a single solution (if it exists) for a given end-effector pose and free angle parameter. If a solution does not exist, it is possible to step through the full range of motion of the redundant joint to *search* for an inverse kinematics solution.

IV. MOTION PLANNING ALGORITHM

The motion planning algorithm we describe in this paper operates by constructing and searching a motion-primitive based graph [5] using pre-defined and runtime-generated motion primitives. The task of the graph search itself is to find a path in the constructed graph, from the state that corresponds to the current configuration of the robot, to any state at which the object is at the desired (goal) location and orientation. In other words, we consider the problem of finding a motion that gets the object grasped by the two arms from its current pose to the goal pose.

In the following sections, we explain all of the components of the algorithm with a major focus on the new graph representation, and the informative heuristic that efficiently guides the search in finding the solution.

A. Graph Construction

Previously, when planning paths for single arm manipulation, we represented the configuration space of the arm in joint space [5]. Thus, if we needed to plan motions for a robot arm with 7 DoF, it would result in a graph with 7 dimensions.The combination of informative heuristics, anytime graph search and adaptive motion primitives addressed the high dimensionality of the statespace. However, if we were to construct a graph in the same way for dual-arm tasks

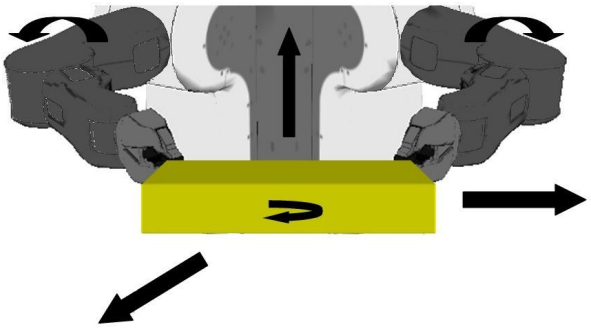
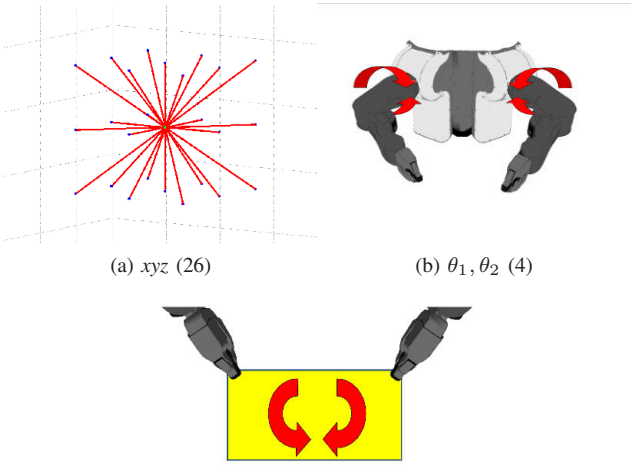


Fig. 2: The six degrees of freedom in the statespace.



(a) *xyz* (26) (b) θ_1, θ_2 (4)

(c) θ_{yaw} (2)

Fig. 3: The set of 32 motion primitives we used.

then we would end up with a 14 dimensional statespace. Fortunately though, the object constraint allows for a more compact graph representation that we will now describe.

The graph is constructed using a lattice-based representation. A lattice is a discretization of the configuration space into a set of states, and connections between these states, where every connection represents a feasible path. Let us use the notation $G = (S, E)$ to denote the graph G we construct, where S denotes the set of states of the graph and E is the set of transitions between the states. The states in S are the set of possible (discretized) 4 DoF poses of the object coupled with the joint angles of one joint (chosen to represent the redundancy) in each arm. That is, we define a state s as a 6-tuple, $(x, y, z, \theta_{yaw}, \theta_1, \theta_2)$ where (x, y, z) describe the global position of the center of the object, θ_{yaw} is the object's global yaw angle and θ_1, θ_2 are the joint positions of the redundant joint in the right arm and the left arm, respectively. Refer to Figure 2 for a visualization of the 6-tuple for the PR2 robot.

The transitions in E are comprised of a set of feasible motion primitives. A motion primitive is defined here as a vector of (translational and yaw) velocities of the object and the two redundant joint velocities. The graph is dynamically constructed by the graph search as it expands states because pre-allocation is infeasible for a 6 dimensional graph.

The set of motion primitives that we used during experimentation can be seen in Figure 3. The set includes 26 motions that just translate the object one cell in the same way as the edges in a 26-connected grid. It also includes two motions that rotate either free angle in both directions. Finally, it includes two motions that just yaw the object in the world frame. We now describe how the motion primitives are used.

Before a successor of state s, s' can be added to the graph it must be checked for feasibility. That is, we check that joint configurations for both arms exist within the joint limits and are collision free. We use an inverse kinematics solver to compute joint configurations for each arm that satisfy the state's coordinates. One reason why the solver may fail is if the solution found violate the joint limits of the arms. If a solution is found for each arm, then the arm configurations are forward simulated and checked for collisions with each other, with obstacles in the environment, and for collisions between the grasped object and the environment.

If the inverse kinematics solver fails to compute a solution for one or both of the arms, then rather than just reject the invalid successor completely, we search over the redundant joint for that arm for a valid solution. If a solution exists, then we generate an adaptive motion primitive [6], or "a primitive that is generated at runtime". Essentially, the motion primitive that was used to reach the invalid state s' is duplicated but with a new value for θ_1, θ_2 or both. This results in a successor state that is the same as s' with the exception of θ_1, θ_2 or both.

It should also be noted, that while we are using a very basic set of motion primitives, more complicated motions could also be used by our approach, e.g. translating the object along an arc through the *xy* plane while rolling the right arm's redundant joint. The set of 32 motion primitives that we described earlier were chosen because they provide a dense coverage of the workspace. In addition, smoother motion primitives can also be used by generating a set of intermediate states for the motion that are not projected onto the discretized grid in the state space. Generating and using such motion primitives is a problem that we intend to pursue further in the future. Using a larger set of primitives may result in smoother paths at the expense of planning time.

We found that this task space representation is advantageous over our previous joint space representation [6] in a couple of ways. A key advantage is the reduced discretization effects in the planned solution. Now that the path length is being optimized in the position space of the object, it allows for multiple joints to be moved at the same time whereas previously, our set of primitives was limiting the trajectory to move one joint at a time giving off a very *robotic* appearance. Another advantage of this representation is that it allows for the heuristic to be more informative and easier for the search to follow. This will be described in detail in the Heuristic

section.

B. Cost Function

The cost function is designed to minimize the path length of the end effector while maximizing the distance between the manipulator and nearby obstacles along the path. The cost of traversing any transition between states s and s' in graph G can therefore be represented as $c(s, s') = c_{cell}(s') + c_{action}(s, s')$. The action cost, c_{action}, is the cost of the motion primitive which is generally determined by the user. The soft padding cost, c_{cell}, is a cost applied to cells close to obstacles to discourage the search from planning a path that drives any part of the manipulator close to nearby obstacles if a safer path is possible. (For example, in our experiments, we set $c_{cell}(s')$ to be twice more expensive than normal cell cost if the configuration of the arm at s' has distance to the nearest obstacle that is less than 20 cm and to be five times more expensive if the distance is less than 10 cm.) In addition, if desired, one can also add an additional term to the cost function to penalize large changes in any particular joint angle to transition from s to s'.

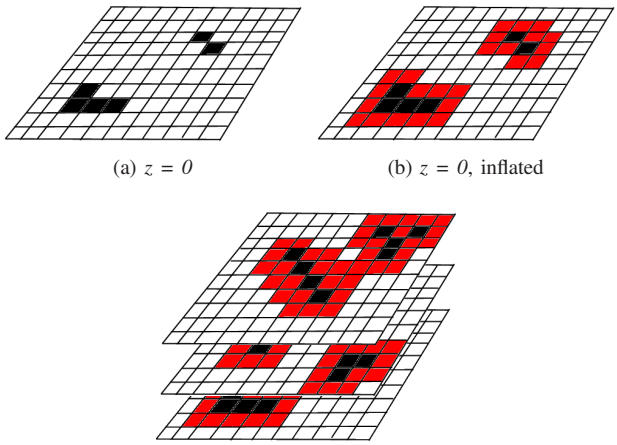
An alternative cost function that we considered minimizes the execution time of the planned trajectory. To do so, the planner is configured with the desired joint velocities for each joint in the arm. Then the cost of a transition between states is computed by solving for the longest joint motion amongst all of the joints and then multiplying its execution time by the desired cost per second. Minimizing execution time usually has the same effect as minimizing the path length of the object, however, tight joint limits can sometimes cause them to not correlate at all. While this cost function offered promising results, it was not used any further in our experiments.

C. Heuristic

Heuristic-based search algorithms such as the graph search used here, depend on informative heuristics to efficiently guide the search in promising directions towards a feasible solution. These algorithms require that the heuristic function is admissible and consistent.

The ability to manipulate objects through cluttered environments is the primary motivation of our research, and so a heuristic function that efficiently circumvents obstacles is necessary. Early in our approach we used a 3D Dijkstra search to compute the cost of the least-cost path from the position of the inner sphere of the object at a given state to the object pose at the goal state. A similar method can be found in [5], however the end effector pose is used in that case. This proved to be an informative heuristic here as well. However, given the additional upright orientation constraint of the problem, we developed a more informative heuristic that exploits this constraint.

Instead of modeling the object as a sphere when performing the 3D Dijkstra search, we can instead model it as a cylinder because we are constraining the object from rolling or pitching. The radius of the cylinder is the radius of the inner circle of the object, i.e. the circle centered at the geometric center of the object (in the *xy* plane) with the largest radius such that it is completely contained within the object footprint, and the height of the cylinder is the height of the object.



(a) $z = 0$ (b) $z = 0$, inflated

(c) $z = 0$, $z = 1$, $z = 2$, inflated

Fig. 4: The obstacles are shown in black and the inflated cells are red. The radius of the inner circle of the object on the *xy* plane is 1 cell. The height of the object is 3 cells so 3 *xy* planes must be checked for collisions.

To compute the heuristic, we first inflate the obstacles on each *xy*-plane by the radius of inner circle of the object by iterating through the *z*-axis of the grid. Then on each call to the heuristic function, *h(x,y,z)*, we check if cells *(x,y,z),(x,y,z+1), ... ,(x,y,z+n)* are collision free, where n is the height of the cylinder in cells. A detailed example can be found in Figure 4.

Modeling the object as a cylinder is significantly more informative than using an inner sphere when the object's dimensions are not similar along each axis, e.g. a tray which is very wide and flat. The heuristic is then capable of guiding the search through tighter spaces, e.g. when manipulating a tray between two shelves of a bookshelf.

We use the radius of the inner circle along the *xy*-plane of the object so that the heuristic is admissible, meaning that it underestimates the cost-to-goal for any full-dimensional state with given (x, y, z) of the object. Needless to say it does not mean that if a feasible 3D path exists from the state to the goal then a feasible motion plan exists to manipulate the object to the goal. It is interesting to note that using the radius of the outer circle may be much more informative when guiding the search especially when the inner and outer circles differ by a large amount. However, using the outer circle to compute heuristics sacrifices the completeness of the planner, i.e., the planner may not find a solution even if one exists. In the Experimental Results section we include a comparison of results from when we used the radius of the inner and outer circles.

It is important to note that the heuristic for a given state is computed when needed unlike in our previous

approaches [6], where the heuristic was precomputed for the entire workspace before planning began. This pre-planning step required up to 0.6 seconds. Instead, now we initially run the Dijkstra search until it expands the start state. Then if the search ever encounters a state which has not yet been added to the Dijkstra search tree, we resume the Dijkstra search until the state of interest is expanded. This way we avoid expanding all of the states in the environment.

D. Search

Any standard graph search algorithm can be used to search the graph G that is constructed. Given the complexity of the graph, however, optimal graph search algorithms such as A* [8] are infeasible to use. Instead we chose an *anytime* version of A* - Anytime Repairing A* (ARA*) [15].This algorithm generates an initial, possibly suboptimal solution quickly and then concentrates on improving this solution while deliberation time allows. The algorithm guarantees completeness for a given graph G and provides a bound ϵ on the suboptimality of the solution at any point of time during the search. ARA* speeds up the typical A* search by inflating the heuristic values by a desired inflation factor, ϵ. An ϵ greater than 1.0 will produce a solution guaranteed to cost no more than ϵ times the cost of an optimal solution in the graph.

V. EXPERIMENTAL RESULTS

Producing a set of randomly generated experiments is a challenging task for dual-arm object manipulation. Kinematic constraints of the arms, the size of the grasped object and the positions and orientations of the grasps result in a very tight feasible workspace, not to mention obstacles. We manually picked start and goal poses for the object, by generating inverse kinematics (IK) solutions corresponding to them and checking that the solutions are collision free. We conducted twelve experiments that were inspired by practical manipulation scenarios in four different cluttered environments with five different objects. All twelve experiments were implemented in simulation first and then on the PR2 robot itself. Figure 5 shows the different simulation environments. The obstacles are in purple and the collision model of the manipulated objects can be seen in cyan. The actual objects that were used during the real robot experiments can be seen in Figure 6.

The results of the simulated experiments are shown in Table I. In all of the runs the planner was initialized with an $\epsilon = 100$ and was given 15.0 seconds to generate a more optimal solution if time permitted. The ϵ of the final solution found is listed in the third column. The planning times include the time it takes to compute the heuristic. The resolution of the object's pose is $2cm$ for the position and 5° for the yaw of the object as well as 2° for both of the redundant joints. All of the tests require that the planner computes a path to a 4-DoF pose constraint for the object with a tolerance of 5° in the final yaw of the object and a $2cm$ tolerance in the position of the object. We do not require the redundant joints to reach the goal at specified joint angles. We used the set of 32 motion primitives shown in Figure 3.

Fig. 5: Clockwise from top left: stick around a pole, wood board in bookshelf, tray with wine glasses under a table, tray with wine glasses near wall and tray with a scotch glass in bookshelf.

Fig. 6: The objects that were modeled for the simulated experiments. The stick is not shown.

Earlier we noted that while the radius of the inner circle of the object in the *xy*-plane is used when computing the heuristic, the search could be sped up by using the radius of the outer circle at the loss of the guarantee on the completeness of the search. Table II shows the results of the same set of experiments but using the radius of the outer circle to compute the heuristic. The statistics do in fact verify the predicted speedup and without failures for this particular set of experiments. We plan to examine the incorporation of this informative heuristic, without sacrificing guarantees on completeness, in greater detail in future work.

A selected set of experiments that were performed on the PR2 robot can be seen in the attached video. During the experiments, the planner was executed onboard the robot. The planning times and number of expansions can be found in Table III. An $\epsilon = 30$ was used to plan and the first solution

TABLE I: Simulation Results (12 trials)

Time until First Soln. (s)	Expands. until First Soln.	ϵ_{final}	Expands. until Final Soln.
0.31	182	3	8,161
0.15	76	3	7,584
0.33	182	3	6,265
2.01	544	5	5,021
1.07	379	4	7,991
0.98	432	4	6,445
14.88	6,773	100	6,785
0.56	31	3	6,714
0.57	34	3	5,960
1.06	322	5	4,932
0.14	62	3	7,344
0.13	68	3	6,437

TABLE II: Simulation Results of Heuristic with radius of the outer circle in the *xy*-plane of the object. (12 trials)

Time until First Soln. (s)	Expands. until First Soln.	ϵ_{final}	Expands. until Final Soln.
0.39	31	2	7,758
0.32	43	2	6,089
0.4	29	2	5,517
0.79	145	3	5,977
0.98	208	3	6,527
0.28	26	2	6,368
2.16	516	3	6,290
0.5	40	3	6,454
0.52	38	3	6,027
12.16	1,996	4	2,961
0.07	38	3	7,015
0.12	68	3	6,703

found was executed. The radius of the inner circle was used to compute the heuristic in all of the runs. The trajectories in the video were generated only by parameterizing (in time) the paths generated by the planner. No smoothing operations were performed on the paths themselves. The short jerky motion at the start of each trajectory does not, to the best of our knowledge, arise from the nature of the planned paths. The long delay in the video was the result of a delay encoded in our experimental system (and not due to long planning times).

The video shows that while the trajectories appear to be reasonably smooth in the position space of the end-effectors, they may not be smooth in the joint space of the arms. While the speedup of the video enhances this effect, another possible reason for this is that we are allowing the IK to search over the range of positions of the redundant joints without applying a cost for rolling either of the redundant joints. If less erratic motion is desired in the upper arm roll joints of the PR2, then an additional transition cost could be used.

As a demonstration of the consistency in the planner's solutions, the video includes two pairs of identical trajectories. The trajectories titled 'Table 1' and 'Table 3' as well as 'Table 2' and 'Table 4' have the same number of expansions

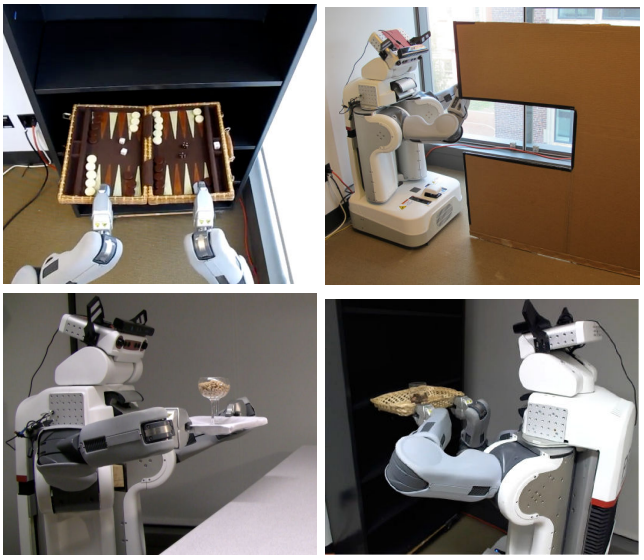


Fig. 7: Shown here are four of the experiments that were run on the PR2. Refer to the video for the execution of eight planning instances in three of these environments.

TABLE III: Planning Times of Attached Video

	Planning Time (s)	Expands.
Backgammon 1	1.03	523
Backgammon 2	0.61	357
Wall 1	9.28	1,710
Wall 2	2.59	619
Table 1	0.38	39
Table 2	0.38	36
Table 3	0.36	39
Table 4	0.34	36

and produce the same exact trajectories but are the result of separate unique calls to the planner. More generally, for planning episodes that occur in similar environments with similar start and goal pairs, the generated motions tend to be similar.

Table IV contains the results of ten trials in which two planners were asked to manipulate a 0.36m x 0.25m x 0.02m tray from above a tabletop to below it and vice versa. The table shows the planning time results of our approach as well as those of a randomized planner (SBL [20]). The tabletop is similar to the one shown in Fig. 5. The planners were given the same exact set of ten unique start-goal pairs. The same set of 32 motion primitives that were used throughout the

TABLE IV: Manipulating a Tray Over-Under Tabletop (10 trials)

	mean time (s)	std dev. time (s)
search-based	0.70	0.47
sbl	2.77	1.55

paper were used in these experiments as well as an $\epsilon = 10$.

VI. CONCLUSION

We presented a search-based motion planning algorithm for dual-arm object manipulation with an upright orientation constraint. In our approach, we leveraged a compact representation of the problem and an informative heuristic that exploits the orientation constraint in creating a method that can efficiently plan dual-arm motions in less than two seconds in over ninety percent of our runs. Our algorithm relies on an anytime graph search to generate initial solutions quickly as well as provide theoretical guarantees on completeness, consistency and provide a bound on the suboptimality of the solution with respect to the graph used to represent the planning problem. We believe that the upright constraint on the object is a very practical one and it covers many of the dual-arm manipulation scenarios found in the home and in other human service tasks. Our experiments on the PR2 showed the ability of the planner to handle real world cluttered manipulation scenarios and its ability to generate motions that can be executed on the robot with minimal post-processing.

VII. ACKNOWLEDGEMENTS

We thank Willow Garage for their partial support of this work. In addition, this research was partially sponsored by the Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0016. We would also like to thank Mike Phillips for his invaluable input.

REFERENCES

[1] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, Alvaro Collet, and James J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, May 2009.

[2] Christoph Borst, Christian Ott, Thomas Wimbock, Bernhard Brunner, Franziska Zacharias, Berthold Baeum, Ulrich Hillenbrand, Sami Haddadin, Alin Albu-Schaeffer, and Gerd Hirzinger. A humanoid upper body system for two-handed manipulation. In *IEEE International Conference on Robotics and Automation*, pages 2766–2767, April 2007.

[3] Matei Ciocarlie, Kaijen Hsiao, E. Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan Alexandru Sucan. Towards reliable grasping and manipulation in household environments. In *ISER*, New Delhi, India, December 2010.

[4] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Planning for autonomous door opening with a mobile manipulator. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[5] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based Planning for Manipulation with Motion Primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[6] Benjamin J. Cohen, Gokul Subramanian, Sachin Chitta, and Maxim Likhachev. Planning for Manipulation with Adaptive Motion Primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. J. Kuffner. Bispace planning: Concurrent multi-space exploration. In *Robotics: Science and Systems*, Zurich, Switzerland 2008.

[8] 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.

[9] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *IEEE International Conference on Robotics and Automation*, May 2010.

[10] Andreas Hormann. On-line planning of action sequences for a two-arm manipulator system. In *IEEE International Conference on Robotics and Automation*, May 1992.

[11] Dov Katz, Emily Horrell, Yuandong Yang, Brendan Burns, Thomas Buckley, Anna Grishkan, Volodymyr Zhylkovskyy, Oliver Brock, and Erik Learned-Miller. The UMass Mobile Manipulator UMan: An Experimental Platform for Autonomous Mobile Manipulation. In *IEEE Workshop on Manipulation for Human Environments*, Philadelphia, USA, August 2006.

[12] Yoshihito Koga and Jean-Claude Latombe. Experiments in dual-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, May 1992.

[13] Yoshihito Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, May 1994.

[14] Tsai-Yen Li and Jean-Claude Latombe. On-line manipulation planning for two robot arms in a dynamic environment. In *IEEE International Conference on Robotics and Automation*, May 1995.

[15] 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.

[16] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *IEEE Intl. Conf. on Robotics and Automation*, 2010.

[17] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[18] 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*, pages 489–494, 12–17 May 2009.

[19] Radu Bogdan Rusu, Ioan A. Şucan, Brian Gerkey, Sachin Chitta, Michael Beetz, and Lydia E. Kavraki. Real-time perception guided motion planning for a personal robot. In *International Conference on Intelligent Robots and Systems*, St. Louis, USA, October 2009.

[20] Gildardo Sanchez and Jean Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *International Symposium on Robotics Research*, 2001.

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

[22] Siddhartha Srinivasa, David Ferguson, Michael Vande Weghe, Rosen Diankov, Dmitry Berenson, Casey Helfrich, and Hauke Strasdat. The Robotic Busboy: Steps Towards Developing a Mobile Robotic Home Assistant. In *Intl. Conference on Intelligent Autonomous Systems (IAS-10)*, July 2008.

[23] 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.