Search-based Planning for Manipulation with Motion Primitives

Benjamin J. Cohen*, Sachin Chitta†, Maxim Likhachev*,
* Computer and Information Science, GRASP Laboratory, University of Pennsylvania, Philadelphia PA 19104
{bcohen,maximl}@seas.upenn.edu
† Willow Garage Inc., Menlo Park 94025, USA
{sachinc}@willowgarage.com

Abstract— **Heuristic searches such as A* search are highly popular means of finding least-cost plans due to their generality, strong theoretical guarantees on completeness and optimality and simplicity in the implementation. In planning for robotic manipulation however, these techniques are commonly thought of as impractical due to the high-dimensionality of the planning problem. In this paper, we present a heuristic search-based manipulation planner that does deal effectively with the high-dimensionality of the problem. The planner achieves the required efficiency due to the following three factors: (a) its use of informative yet fast-to-compute heuristics; (b) its use of basic (small) motion primitives as atomic actions; and (c) its use of ARA* search which is an anytime heuristic search with provable bounds on solution suboptimality. Our experimental analysis on a real mobile manipulation platform with a 7-DOF robotic manipulator shows the ability of the planner to solve manipulation in cluttered spaces by generating consistent, low-cost motion trajectories while providing guarantees on completeness and bounds on suboptimality.**

I. INTRODUCTION

Many planning problems in robotics can be represented as finding a least-cost (or close to least-cost) trajectory in a graph. Heuristic searches such as A* search [6] have often been used to find such trajectories. There are a number of reasons for the popularity of heuristic searches. First, most of them typically come with strong theoretical guarantees such as completeness and optimality or bounds on suboptimality [14]. Second, there exist a number of anytime heuristic searches that find the best solution they can within the provided time for planning [5], [18], [19], [12] . Third, there exist a number of incremental heuristic searches that can re-use previous search efforts to find new solutions much faster when previously unknown obstacles are discovered [16], [9]. Finally, treating a planning problem as finding a good quality path in a graph is advantageous because it allows one to incorporate complex cost functions, complex constraints and represent easily arbitrarily shaped obstacles with grid-like data structures [4], [11]. Consequently, heuristic search-based planning has been used to successfully solve a wide variety of planning problems in robotics.

Despite the wide popularity of heuristic searches, they typically have not been used for motion planning for high-DOF robotic manipulators [2]. The main reason for this is the high-dimensionality of the planning problem. In this paper, we present a heuristic search-based planner for manipulation that combats effectively this high dimensionality by exploiting the following three observations. First, the solutions found in a low-dimensional manifold of the workspace can serve as highly informative heuristics and can therefore guide search in the joint angle state-space quite well. Second, the majority of complex motion plans can be decomposed into a small set of basic (small) motion primitives. These motion primitives can be used to construct a graph on which the heuristic search is executed. The graph is sparser compared with the N-dimensional grid resulting from the discretization of N joint angles for an N-DOF manipulator. As explained later, this motion primitive-based graph also allows us to optimize for smoothness in actions at a small increase in the size of the graph. Third, while finding a solution that is *provably* optimal is expensive, finding a solution of *bounded suboptimality* can often be drastically faster. To this end, we employ an anytime heuristic search, ARA* [12], that finds solutions with provable bounds on suboptimality and improves these solutions until allotted time for planning expires.

The paper is organized as follows. It first briefly describes some of the existing approaches to planning for manipulators including the widely popular sampling-based approaches. It then explains our heuristic search-based planner including the graph it builds, the heuristic it uses and the search it employs. Section IV mentions other extensions and optimizations we have developed for the planner including the extension that allows the planner to manipulate objects while satisfying constraints (e.g., carrying a cup without turning it upside down). Section V presents the experimental analysis of the planner in simulation and on a real mobile manipulation platform with a 7-DOF robotic manipulator (Figure 1). The experimental results shows the ability of the planner to solve manipulation in cluttered spaces by generating consistent, low-cost motion trajectories while providing guarantees on completeness and bounds on suboptimality.

II. RELATED WORK

Sampling-based motion planners [8], [10], [1] have gained tremendous popularity in the last decade. They have been shown to consistently solve impressive high-dimensional motion planning problems. In addition, these methods are simple, fast and general enough to solve a variety of motion planning problems. Sampling-based methods have also been extended to support motion constraints through rejection sampling [17].

Fig. 1: The PR2 mobile manipulation platform.

Our approach to motion planning differs from these algorithms in several aspects. First, sampling-based motion planner are mainly concerned with finding any feasible path rather than minimizing the cost of the solution. By sacrificing cost minimization, these approaches gain very fast planning speeds. Searching for a feasible path however may often result in the solutions of unpredictable length with superfluous motions, motions that graze the obstacles, and jerky trajectories that may potentially be hard for the manipulator to follow. To compensate for this, various smoothing techniques have been introduced. While often helpful, they may fail to help in cluttered environments. Second, sampling-based approaches provide no guarantees on the sub-optimality of the solution and provide completeness only in the limits of samples. In contrast, the search-based planning tries to find the solutions with minimal costs and provides guarantees on solution suboptimality w.r.t. the constructed graph. These aspects are valuable when solving motion planning problems for which the minimization of objective function is important and when consistent behavior is expected

Several motion planning algorithms have been developed that also try to find solutions of minimal cost [7], [15]. One of the most recent algorithms in this category is Covariant Hamiltonian Optimization and Motion Planning, or CHOMP [15]. It works by creating a naive initial trajectory from the start position to the goal, and then running a modified version of gradient descent on the cost function. CHOMP offers numerous advantages over sampling-based approaches such as the ability to optimize trajectories for smoothness and to stay away from obstacles when possible. Our approach is similar to CHOMP in that we also recognize the importance of cost minimization but, in addition, we provide the guarantees on the global solution suboptimality.

III. ALGORITHM

The operation of our algorithm is based on constructing a motion primitive-based graph and searching this graph for a low-cost solution. In the following sections we explain the construction of the motion primitive-based graph, the cost function used to assign edgecosts in the graph, the heuristic function that guides the graph search in finding the solution, and finally graph search itself. It is important to note that the actual graph construction is interleaved with graph search so that only the portion of the graph needed by search is actually stored in memory. This is important because the full graph is too large to store in memory and would be too computationally expensive even to construct.

The task of the graph search itself is to find a path in the constructed graph from a state that corresponds to the current configuration of the manipulator to a state for which the pose of the end-effector satisfies the goal conditions. In other words, we consider the problem of finding a motion that gets the manipulator from its current configuration to *any* configuration with the end-effector at the desired pose.

A. Graph Construction

The graph structure we use was inspired by the success of lattice based planners in planning dynamically feasible trajectories [11]. Lattice-based representation is a discretization of the configuration space into a set of states, and connections between these states, where every connection represents a feasible path. As such, lattices provide a method for motion planning problems to be formulated as graph searches. However, in contrast to many graph-based representations (such as 4-connected or 8-connected grids), the feasibility requirement of lattice connections guarantees that any solutions found using a lattice will also be feasible. This makes them very well suited to planning for non-holonomic and highly-constrained robotic systems. 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) joint configurations. Similary to lattice-based representations, we confine the transitions in E to be a predefined set of feasible motion primitives. [1] We define a state s as $n+1$-tuple $(\theta_0, \theta_1, \theta_2, ..., \theta_n, m)$ for a manipulator with n joints. In this definition, m represents the index of the motion primitive used to reach the state s. This additional variable is used for path smoothing and will be explained later in the paper. The graph is dynamically constructed by the graph search as it expands states since pre-allocation of memory for the entire graph would be infeasible for an n-DOF manipulator with any reasonable n. Each motion primitive is a single vector of joint velocities, $(v_0, v_1, v_2, ..., v_n)$ for all of the joints in the manipulator. The set of primitives is the set of the smallest possible motions that can be performed at any given state. Therefore, a primitive is the difference in the global joint angles of neighboring states. This set is the same for any state at which it is executed which allows us to pre-evaluate and pre-compute motion primitives offline.

[1]The term “motion primitive” is sometimes used in planning literature to represent a higher level action such as opening a door, swinging a tennis racket, or pushing a button. This is different from our use of the term: we use “motion primitive” to denote a basic (atomic) feasible motion.

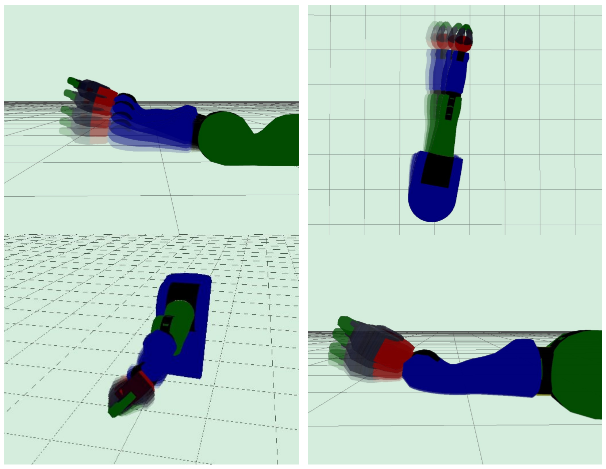


Fig. 2: Four motion primitives are shown here. Each one of these primitives moves one of the joints. From the top left to the bottom right, elbow flex, shoulder pan, forearm roll and wrist pitch.

In our experiments with a 7-DoF manipulator, 14 basic motion primitives were used along with eight additional compound primitives. Seven basic motion primitives moved one joint each a pre-defined amount in the positive direction and the other seven moved the same amount in the negative direction. The eight additional motion primitives moved several joint angles at a time and provided better coverage of the workspace based on the arm's kinematics and joint limitations.

The algorithm we are proposing in this paper constructs the proper framework needed to use a more complex set of motion primitives. As stated above, while some of the primitives used during our experimentation comprised of multi-dimensional motion, most of them were defined by a motion in only one joint. This was done for the sake of simplicity during the research phase. We are currently researching how to design more complex motion primitives that explore the workspace of the manipulator with even coverage and in a more linear fashion.

B. *Cost Function*

The cost function is designed to minimize the path length, maximize path smoothness and maximize the distance between the manipulator and the obstacles around it. The cost of traversing any transition between states s and s' in graph G can therefore be represented as $c(s,s') = c_{cell}(s') + w_{action} * c_{action}(s,s') + w_{smooth} * c_{smooth}(s,s')$. The latter cost terms are weighted with w_{action} and w_{smooth}. The weights can be chosen by the user to govern the amount of smoothness desired. A larger w_{smooth} results in a smoother path at the expense of optimality in the length of the path.

1) $c_{cell}(s')$: The $c_{cell}(s')$ term is computed by calculating the shortest distance between the manipulator at state s' and the nearest obstacle. In our experiments, a second process was dedicated to computing a distance field (a distance to the nearest obstacle for each 3D voxel) for each updated collision map that is constructed from sensory data. On a planning request, the planner would then fetch the most recent distance field and collision map to use. The distance between the manipulator at state s' and the nearest obstacle can then be computed by iterating over all the 3D voxels that the manipulator intersects and finding a voxel with the minimum distance to an obstacle.

2) c_{action}: The action cost is used to assign a fixed cost for each motion primitive. In our experiments, we assigned uniform c_{action} costs. In other domains however, one may assign different motion primitive costs based on various things such as power consumption, size of the links being rotated or translated or as a means to favor moving some joints instead of others if possible.

3) $c_{smooth}(s,s')$: The smoothness cost is used to penalize edges in the graph that correspond to choppy motions in the trajectory. One way to implement this cost would be to add the velocity of each joint to each state in the graph G. Then, for a 7 DoF manipulator, instead of planning in a 7 dimensional statespace, $(\theta_1, \theta_2, ...\theta_7)$, we would have to plan in a 14-dimensional state-space, that is, each state would be defined as $(\theta_1, v_1, \theta_2, v_2, ..., \theta_7, v_7)$. This is a dramatic increase in the dimensionality of the problem.

As described in the previous section, our solution to this is to augment each state with a single variable m which represents the index of the motion primitives that connects the previous state with the current. This is possible because of our use of a fixed motion primitive set. The smoothness cost, $c_{smooth}(s,s')$, is a cost applied to the change in velocities between states s and s'. The magnitude of the change in velocities can be represented by, $\sum_{i=0}^{n}(v_i(s) - v_i(s'))^2$ where v_i are the joint velocities. The joint velocities of a state are determined by the motion primitive index m.

C. *Heuristic*

The purpose of a heuristic function is to improve the efficiency of the search by guiding it in promising directions. A common approach for constructing a heuristic is to use the results from a simplified search problem (e.g. from a lower-dimensional search problem where some of the original constraints have been relaxed). For a heuristic function to be most informative, it must capture the key complexities associated with the overall search, such as mechanism constraints or the environment complexities. Heuristic-based search algorithms require that the heuristic function, h, is admissible and consistent. This is true when $h(s_{goal}) = 0$ and for every pair of states s, s' such that s' is an end state of a single action executed at state s, $h(s) \leq c(s,s') + h(s')$, where $h(s)$ is a heuristic of state s, s_{goal} is a goal state (any state with the end-effector in the desired pose) and $c(s,s')$ is the cost of the action that connects s to s'. The planner that we are proposing performs a graph search in a state-space defined by the joint angle configurations while the goal pose is defined by the position and orientation of the end effector. It is therefore necessary to use a heuristic function that is informative about the end effector position and orientation, (x,y,z,r,p,y). To this end, we compute the heuristic of a

heuristic	expansions	planning time(sec)	solution cost
dijkstra	88,858	13.65	46000
euclidean	432,561	60	-

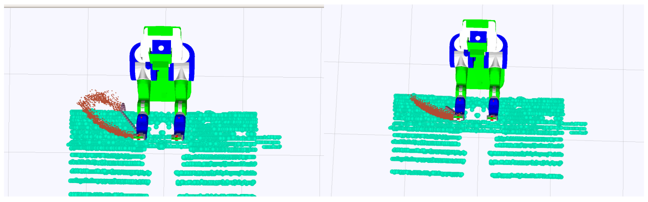
TABLE I: Planning statistics with an $\epsilon = 5$. The trial with Euclidean distance as the heuristic function did not return a solution within 60 seconds so the trial was terminated.

state s as $h(s) = h_{xyz}(s) + w * h_{rpy}(s)$, where h_{xyz} estimates the cost to the desired end-effector position, h_{rpy} estimates the cost to the desired end-effector orientation and w is the weight of the latter component.

1) h_{xyz}: The ability to plan robustly in cluttered environments is the primary motivation of this research, and so a heuristic function that efficiently circumvents obstacles is necessary. Simplifying a planning problem by removing its complexity and removing some dimensionality is a standard technique in creating an informative heuristic function. In the same vein, we use a 3D Dijkstra's search to find the costs of the least-cost paths from every 3D voxel to the 3D voxel that corresponds to the goal position (x, y, z) of the end-effector while avoiding obstacles. During the full planning, the heuristic component $h_{xyz}(s)$ for any state s is computed as follows: we first compute the coordinates of the end-effector of the manipulator configuration defined by state s; we then return the cost-to-goal computed by the 3D Dijkstra's search for the voxel with these coordinates. The h_{xyz} heuristic proves to be an informative heuristic in directing the graph search around obstacles in a cluttered workspace. It is important to note though that depending on the configuration of the obstacles in the environment, it is possible that the path to the goal computed by the Dijkstra's search may not be reachable by the manipulator. In these situations, h_{xyz} can be uninformative. Table I shows some statistics comparing the effect of Dijkstra's-based heuristics and Euclidean distance-based heuristics on a single planning run through cluttered environment. It shows that the use of Dijkstra's search can easily make a drastic impact on the performance of the planner.

Dijkstra's algorithm is helpful in guiding the search through cluttered environments. If the planner is being used in an application with fairly obstacle free environments, then using the Euclidean distance between the end effector position and the goal will suffice as a heuristic. In our experimentation, we perform a Dijkstra's algorithm on a 100x100x100 grid before every search. The execution of Dijkstra's takes approximately 0.4 seconds to complete. Single Dijkstra's search computes the cost-to-goals for all the cells in the grid and therefore provides all the required h_{xyz} values.

2) h_{rpy}: In order to achieve the proper end effector orientation at the goal pose, we use a heuristic function that computes a cost representing the difference in the orientation of the end-effector at the state in question and the desired orientation of the end-effector. One effective parameterization for describing the difference between the end effector orientation of state s and the orientation of s_{goal} is through the axis-angle representation of a rotation [13]. Thus, $h_{rpy}(s)$ is equal to the angle of rotation about a fixed axis specified by the axis-angle representation of the rotation between the end effector orientation of state s and s_{goal}.


(a) $h_{Dijkstra}$: bird's eye view (b) $h_{Euclidean}$: bird's eye view

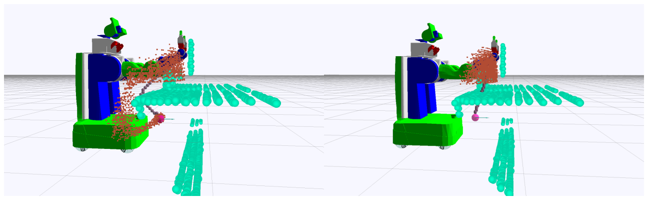

(c) $h_{Dijkstra}$: side view (d) $h_{Euclidean}$: side view

Fig. 3: In the experiment above, the right arm of the PR2 is initially stretched out over a table and a path is desired to a goal pose below the table (represented by the purple sphere with the blue arrow). The teal spheres represent obstacles in the perceived collision map and the brown cuboids represent the end effector position of the expanded states during the search. The purple pipe represents the heuristic function's suggestion for the end effector's path. Dijkstra's algorithm is used as the heuristic in the images on the left and Euclidean distance is the heuristic function used on the right.

D. Search

Any standard graph search algorithm can be used to search the graph G that we construct. Given its size, however, optimal graph search algorithms such as A* [6] are infeasible to use. Instead we employ an anytime version of A* - Anytime Repairing A* (ARA*) [12].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. This is effective at rushing the graph traversal towards the goal state at the cost of solution optimality. An ϵ greater than 1.0 will produce a solution guaranteed to cost no more than ϵ times the cost of an optimal solution. If the path is found within the time allotted, then a followup search is executed with a lower ϵ weight applied to the heuristic. The search is continuously repeated while decrementing the epsilon with every iteration, until either the search time is up or ϵ has reached a value of 1. In doing so, ARA* gains an additional efficiency by

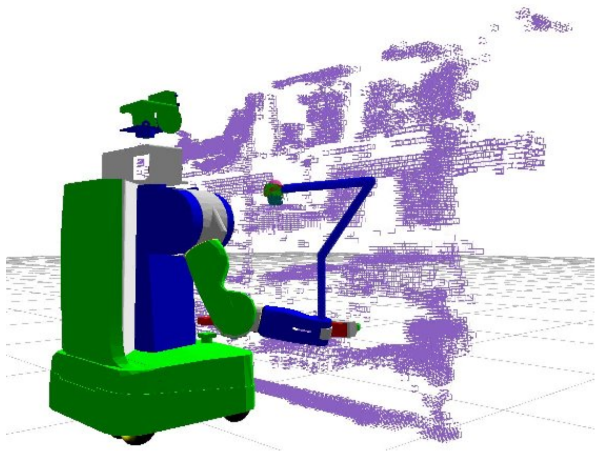


Fig. 4: Visualization of the planning process. The purple squares represent the collision map, The blue pipe represents the heuristic function's suggestion for the end effector's path. The green squares represent the end effector position and orientation of the state expanded during the search.

not re-computing the states that it already computed in its previous search iterations.

Finding a motion that moves the end-effector precisely to the 6-DOF cartesian pose can often make the job of the search very difficult. The looser the tolerance on the desired end-effector pose, the easier it is for the search to find the solution. To simplify the job of the search without sacrificing the precision, we employ inverse kinematics function on the states with end-effector poses within some region around the desired end-effector pose. More precisely, suppose search generates a state s for which the x, y, z coordinates of the end-effector are within small distance (i.e., 15cm) from the goal coordinates of the end-effector. We then seed IK with the full manipulator configuration defined by s and run it to obtain a full configuration for the desired 6 DOF end-effector pose. If IK returns a valid solution, we then check for collisions and joint limits a linearly interpolated path between the configuration defined by s and the configuration returned by IK. If the path is valid, then state s is deemed as a goal state.

IV. EXTENSIONS & OPTIMIZATIONS

A. Features

1) Path Constraints: Many motion planning tasks not only require that the end effector ultimately finds its way to the goal pose, but also require that the manipulator adheres to certain constraints along the way. A common example of a planning task that requires path constraints is picking up a glass filled with liquid and putting it elsewhere. Manipulating a filled glass requires that the end effector remains upright throughout the whole manipulation so as to not spill the glass's contents. Path constraints such as an upright end effector (roll & pitch angles of zero) can be taken into consideration when planning with this planner. Path constraints can be expressed as a bounds on the position or orientation of a specific joint or link. Path constraints can also be defined as a desired joint position of a specific joint. All path constraints are implemented as validity checks on all expanded nodes. During the expansion of a node, if one of its successor nodes does not meet the specified constraints, then it is considered invalid and not placed in the open list. It is common that path constraints can provide a decent speedup to the planner by effectively shrinking the statespace.

2) Multiple Goals: Another feature of the motion planner we are presenting is the ability to handle multiple goal poses as input and return a path to the goal pose with the lowest path cost overall. This can prove useful when a grasp planner finds multiple feasible grasp poses for manipulating an object. The motion planner will compute a path to the grasping pose with the least cost path.

B. Additional Path Smoothing

In addition to the smoothing cost mentioned above, where we apply a cost to change joint velocities along graph edges, the planned path is passed through a short circuit smoothing function [3]. The short circuit smoothing function iterates through the waypoints in the path and tries to connect each waypoint to the furthest waypoint in the path in which there is a direct collision free path between them. If a direct path exists, then the intermediate waypoints are removed from the path and the algorithm continues with the next waypoint.

Smoothing is performed solely to remove the discretization artifacts in the trajectory. The artifacts stem from the discretization of the joint angles in the motion primitives. Unlike sampling-based planning, when given enough time the graph search presented in this paper is guaranteed to find a least-cost motion with respect to the defined discretization. Thus, smoothing to remove inefficient and unnecessary motions is not required. Smoothing however *is* beneficial for removing the discretization artifacts in the motion.

C. Optimizations

In our experimentation, we planned on a voxel grid with one centimeter resolution because we were limited by the resolution of the collision map received from the laser scanner. This accuracy was adequate when planning to objects during tabletop manipulation. To accelerate our collision checking function, we used an occupancy grid with 2 cm resolution to check if a successor state was valid. Thus, two occupancy grids were maintained at all times, allowing the planner to use the higher resolution grid to check if the search has reached the goal, and the lower resolution grid for collision checking.

Once the planner receives the goal state from the user, the planner computes online the cost-to-goals for all the 3D voxels in a discretized 3D obstacle map using Dijkstra's search (as explained in the section on heuristics). We can speed up this operation by using a lower resolution grid for these heuristics computations. The twice lower resolution grid finishes in a third of the time demanded by the larger grid used for planning.

V. EXPERIMENTAL RESULTS

A. Simulation

The algorithm was tested in simulation using an open source simulator called Gazebo. Gazebo is a three dimensional simulator that can demonstrate the dynamics and limitations of the modeled manipulator using the Open Dynamics Engine. The robot model used in simulation is a fairly accurate representation of the `PR2`, the robot built by Willow Garage, that we eventually used for our experimentation. The simulated robot contains all of the sensors used by the actual robot. An open source software framework called `ROS`[2] was used for inter-process communication. The `PR2` is described in greater detail in the following section.

	avg	std	conf. int.
expansions	16580.120	32957.950	5406.48
cost	85698.400	25966.624	3516.36
time	3.499	6.986	0.764

TABLE II: Simulation results for randomly generated test environments. The planner was executed with ϵ =15. The confidence intervals are for 95 $\%$ intervals.

Table II shows one set of experimental results we obtained in simulation. We generated 470 test environments with kinematically feasible 6-DOF desired end-effector positions (goals). In each case, we randomly placed between 3 and 9 cubic obstacles in the workspace of the manipulator. The dimensions of the cubic obstacles ranged randomly in between 10cm and 20cm. Of the 470 test cases, in 68% of them, the planner successfully computed feasible paths within a 40 second time limit. We set a time limit of 40 seconds for each execution of the planner because it was unknown whether all of our generated test cases were feasible or not. The statistics shown in table II were computed based on the calls to the planner that completed within that time frame.

	avg time (sec)	% completed
3 DoF	5.06	100
6 DoF	0.32	33

TABLE III: Simulation results for a cluttered tabletop manipulation scenario. Twelve trials with unique start configurations and goal poses were randomly generated and tested for feasibility prior to the simulation. The planner was executed with $\epsilon = 10$

In addition to the randomly generated environments, we also generated twelve test cases representing a tabletop manipulation scenario. We ran the trials twice. The first set of tests required the planner to plan a path to a goal state defined as an x, y, z coordinate of end-effector. We then ran the same trials again but planned to a full 6-DoF cartesian pose of an end-effector. The `PR2` was placed in front of a table with clutter on it and without much clearance between the shoulder and the edge of the table. Seven of the twelve test cases required the manipulator to bring the end effector from above the table to below and vice versa. The results are given in Table III. They show that the planner is much more effective in achieving a 3-DOF pose (x, y, z) than in achieving a 6-DOF pose. We believe that the reason for it is that our heuristics are not informative enough with respect to the desired orientation. The design of heuristics that are informative with respect to the goal end-effector orientation is one of the promising directions for future work.

B. `PR2` *Experiments*

Experiments were also conducted on a mobile manipulation platform called the `PR2`. The PR2 is a mobile robot designed for mobile manipulation, which makes it an appropriate test bed for our motion planner.

The robot has substantial processing power on board, with two eight core computers available. For sensing, it has six cameras as well as two Hokuyo laser scanners, one of which is mounted on a servo that provides a tilt scanning ability. In our experiments we decided to use the tilt laser scanner for perception of the environment. The tilt laser scanner provided the robot with a dense point cloud of the environment with 270 degrees of coverage around the sensor. Since manipulation generally occurs within one meter of the torso, the accuracy provided by the laser at that distance was quite good. The only drawback to using the tilt scanner is that a full scan of the environment takes around four seconds to complete a full sweep.

The PR2 has two arms with seven degrees of freedom each. However, the specific robot that we used for testing is an experimental model with just one arm. Each arm has a payload of two Kg and enough torque in the wrist to accomplish household chores. The seven joints in the arm are shoulder pan, shoulder lift, upper arm roll, elbow flex, forearm roll, wrist pitch and wrist roll. The gripper has an additional degree of freedom (a two fingered claw) that we did not take into account during planning. The wrist roll and forearm roll joints are continuous joints. The joint limitations of the arm are taken into account when planning.

`ROS` was the software framework used for development, communication and interfacing to the hardware on the `PR2`. The software framework for the `PR2` includes a controller library that provides an interface to a set of trajectory controllers for the different parts of the robot including the arms. The planner interfaces to the controllers using a `ROS` service call[3]. The motion planner was implemented with an interface using a service that gets called when a motion plan for the arm is desired. The experiments conducted on the robot involved planning from different start positions to a set of goal positions in relatively cluttered environments. Snapshots of these experiments are shown in Figure 5.

VI. CONCLUSIONS

We have presented a heuristic search-based manipulation planner. We have shown that by using a motion primitive-based graph coupled with informative heuristics and anytime

[2]`http://www.ros.org`

[3]A `ROS` service call is similar to a Remote Procedure Call (RPC)

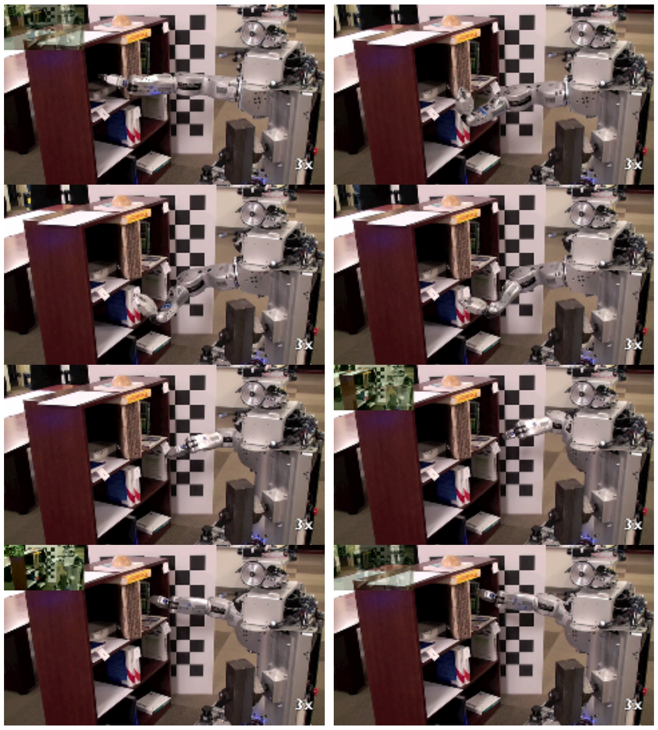


Fig. 5: Snapshots of the robot planning to a goal position in a cluttered environment.

graph search, the planner can deal effectively with the high-dimensionality of the problem. In addition to its explicit cost minimization, the approach provides strong guarantees on completeness and suboptimality. Our experimental analysis on a real mobile manipulation platform with a 7-DOF robotic manipulator show that the planner can frequently solve manipulation in cluttered spaces by generating consistent (i.e., same across different runs), low-cost motion trajectories while providing guarantees on completeness and bounds on suboptimality. While the planner still lacks the efficiency of sampling-based approaches, we believe that our results coupled with the guarantees on sub-optimality are encouraging. In the future, we intend to explore other, more sophisticated, approaches to generating and possibly learning online motion primitives. We would also like to develop other informative heuristic functions for the mobile manipulation tasks.

VII. ACKNOWLEDGMENTS

This work was supported by Willow Garage. We would like to thank the members of the Willow Garage development team for their help.

REFERENCES

[1] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation, VOL.1*, 2007.

[2] Pang C. Chen and Yong K. Hwang. Sandros: A dynamic graph search algorithm for motion planning. In *IEEE Transactions on Robtics and Automation, VOL. 14*, 1998.

[3] Howard Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. In *Principles of Robot Motion*, pages 203–215. Cambridge, MA: MIT Press, 2005.

[4] A. Kanehiro et al. Whole body locomotion planning of humanoid robots based on a 3d grid map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[5] D. Furcy. *Chapter 5 of Speeding Up the Convergence of Online Heuristic Search and Scaling Up Offline Heuristic Search*. PhD thesis, Georgia Institute of Technology, 2004.

[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] Nazareth Bedrossian Jeff M. Phillips and Lydia E. Kavraki. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2004.

[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] S. Koenig and M. Likhachev. Incremental A*. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS) 14*. Cambridge, MA: MIT Press, 2002.

[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)*, 2000.

[11] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research (IJRR)*, 2009.

[12] 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.

[13] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

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

[15] 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.

[16] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1652–1659, 1995.

[17] Ioan Alexandru Sucan and Lydia E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *International Workshop on the Algorithmic Foundations of Robotics*, 2008.

[18] R. Zhou and E. A. Hansen. Multiple sequence alignment using A*. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002. Student abstract.

[19] R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 90–98, 2005.