Planning for Manipulation with Adaptive Motion Primitives

Benjamin J. Cohen*, Gokul Subramanian*, Sachin Chitta†, Maxim Likhachev‡
* Computer and Information Science, GRASP Laboratory, University of Pennsylvania, Philadelphia PA 19104
{bcohen,gosub}@seas.upenn.edu
† Willow Garage Inc., Menlo Park 94025, USA
sachinc@willowgarage.com
‡ Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
maxim@cs.cmu.edu

***Abstract*— In this paper, we present a search-based motion planning algorithm for manipulation that handles the high dimensionality of the problem and minimizes the limitations associated with employing a strict set of pre-defined actions. Our approach employs a set of adaptive motion primitives comprised of static motions with variable dimensionality and on-the-fly motions generated by two analytical solvers. This method results in a slimmer, multi-dimensional lattice and offers the ability to satisfy goal constraints with precision. To validate our approach, we used a 7DOF manipulator to perform experiments on a real mobile manipulation platform (Willow Garage's PR2). Our results demonstrate the effectiveness of the planner in efficiently navigating cluttered spaces; the method generates consistent, low-cost motion trajectories, and guarantees the search is complete with bounds on the suboptimality of the solution.**

I. INTRODUCTION

Heuristic searches such as A* search [6] have often been applied to motion planning problems in robotics. Heuristic searches offer strong theoretical guarantees such as completeness and optimality or bounds on suboptimality [11]. When quicker planning times are required, anytime heuristic searches can be used to find the best solution possible given a time constraint [5], [13], [14], [10]. Heuristic searches are advantageous because they allow the incorporation of complex cost functions and complex constraints while easily representing arbitrarily shaped obstacles as grid-like data structures [3], [9].

Our previous work on search-based planning for manipulation [2], has shown that it is possible to systematically construct a graph and search it with an A* type search, despite the high dimensionality of the motion planning problem. The algorithm used a small set of basic motion primitives to construct a graph on which, every path between nodes represents a kinematically feasible trajectory that the manipulator can follow. To search the graph for a solution, the algorithm used anytime heuristic search, ARA* [10] in conjunction with informative heuristics that account for environmental constraints. While finding an optimal solution is expensive, solutions with provable bounds on the suboptimality can often be found drastically faster using an anytime search.

The algorithm was a step towards applying the strong theoretical guarantees heuristic search offers to planning for manipulation. However, the experimental results showed the algorithm's inability to satisfy 6DOF goal constraints efficiently and accurately. (We define these 6DOF constraints for the end effector goal to be a a 6-tuple (x, y, z, r, p, y) consisting of the 3D Cartesian position (x, y, z) and orientation (roll, pitch, yaw) in the robot's base frame). These inefficiencies were caused by the high dimensionality of the state-space, as well as the restrictions imposed by limiting the construction of the graph to a pre-defined set of motion primitives. In this paper, we present the use of *adaptive motion primitives* (or alternatively *runtime motion primitives*, primitives that are generated on-line) to effectively deal with the inefficiency and inaccuracy of the search. Adaptive motion primitives assist the pre-defined set, through the use of two key additions. First, to combat the inaccuracies of a predefined set of basic primitives, we employ a combination of two analytical solvers that generate the on-the-fly motions necessary to reach the goal constraints. Second, when planning for high degree of freedom manipulators, we can exploit the fact that not all of the joints are necessary to reach the position constraint. Therefore, we propose the use of variable dimensionality in the set of pre-defined motion primitives, to improve the efficiency of the search. Our experimental results confirm that adaptive motion primitives are capable

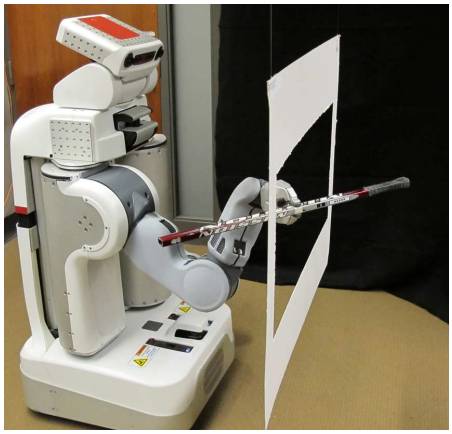

Fig. 1: Manipulating objects in cluttered environments is the primary motivation of this work.

of planning to 6DOF goal constraints more efficiently and more precisely.

The paper is organized as follows. First, it briefly describes some of the existing approaches to motion planning for manipulation, including sampling-based methods. It then explains the core components of our algorithm including the construction of the graph, the heuristic, and the search. In this section, we also present a new heuristic function which we combine with our previously developed heuristics to assist in coping with the previously ignored kinematic complexities of the manipulator. Section IV presents the experimental analysis of the planner in a variety of common manipulation environments as well as a description of our testing on the PR2 robot. These experimental results, display the benefits of employing some or all of the adaptive motion primitives. While our experiments specifically involve the PR2, all the theoretical results of our work carry over to any robot with similar kinematic arm structure.

II. RELATED WORK

The most common approach to motion planning for manipulation are sampling-based methods such as RRT and PRM [7], [8], [1]. They are simple, fast and have been shown to consistently solve impressive high-dimensional planning problems.

There are a few key differences between our approach and sampling based approaches. First, the paradigm underlying sampling based planners does not, by itself enforce any form of solution optimization. Searching for a feasible path may often result in solutions of unpredictable length, with superfluous motions, and motions that graze the obstacles. While trajectory smoothing techniques are helpful, they may fail to help in cluttered environments. Recently developed RRT* [4] provides guarantees in the limit of samples on completeness and the optimality of the solution. Second, sampling based planners do not generate consistent solutions due to randomization. In contrast, graph search-based planning tries to find solutions with minimal cost and provides guarantees on solution suboptimality and completeness (under the constraints of state space and action space). And as with any deterministic graph search-based planning, our approach provides consistency in the solutions: similar solutions are found for similar scenarios.

CHOMP [12], is a method of trajectory optimization that 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 a few major 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 guarantees on the global solution suboptimality.

Last year we presented a novel search-based motion planner for manipulation that used motion primitives to systematically construct a graph and search it with an anytime algorithm [2]. The experimental results showed that planning was feasible but lacked efficiency, especially for 6DOF goals. Planning to 6DOF goal constraints even often failed at returning solutions within a few minutes. A large tolerance on the goal orientation was needed because the search was limited to a pre-defined set of primitives that struggled to reach an arbitrary orientation. In this paper, we introduce the use of adaptive motion primitives. As our experimental results show, this substantially improved the efficiency of planning to 6DOF goal constraints while maintaining the same strong theoretical guarantees on the completeness of the search, consistency in the solution and guarantees on the solution suboptimality, given the state space and action space constraints.

III. ALGORITHM

The algorithm we describe in this paper operates by constructing and searching a graph [2] based on predefined and dynamically created motion primitives of varying dimensionality. The graph search must use the constructed graph to find a path 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 6D pose.

In the following sections, we explain the graph construction, focusing on adaptive motion primitives, the cost function used to assign edge costs in the graph, the heuristics that guide the graph search in finding the solution, and finally the graph search itself.

A. Graph Construction

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) joint configurations and the transitions in E are a set of feasible motion primitives. A motion primitive is the difference in the global joint angles of neighbouring states. We define a state s as an n-tuple $(\theta_1, \theta_2, ..., \theta_n)$ for a manipulator with n joints. A motion primitive is defined as a vector of joint velocities, $(v_1, v_2, ..., v_n)$ for a subset or for all n joints. The graph is dynamically constructed by the graph search as it expands states, as pre-allocation of memory for the entire graph would be infeasible for an n-DOF manipulator with any reasonable n. We now present three different types of motion primitives that connect state s to its succeeding states, $succ(s)$.

1) Static Motion Primitives with Variable Dimensionality: Planning in a high dimensional lattice is computationally expensive and requires a lot of system resources. An important observation, however, is that when planning for manipulation with a high dimensional manipulator, not all of the available degrees of freedom are actually needed to find a safe path to

the goal region or even to the goal position itself. Frequently, using a subset of the joints is fully adequate for computing a feasible path to the vicinity of the desired end-effector pose. The reason is that there are many more choices for the motion of the arm when it is constrained to a particular end-effector pose. In the vicinity of the goal pose, the planner may have to exercise other joints such as the wrist joint to satisfy certain orientation constraints.

This observation motivated us to generate a multi-dimensional set of static motion primitives. A subset of motion primitives can be used to quickly search for a path to the goal region. These motion primitives are chosen with the goal of achieving a lower-dimensional state-space. Once the search enters a potentially cluttered goal region, the planner uses the complete set of full dimensional primitives to search for a path to the goal pose in a full-dimensional state-space. The end result is a more efficient search through a multi-dimensional lattice.

We define MP_{lowD} to be a subset of the predefined set of primitives such that each can only change a subset of joints. This means that in the regions where only the motion primitives from MP_{lowD} are used, the state-space is lower-dimensional (its dimensionality is the number of joints that are modified by the motion primitives that are in MP_{lowD}). MP_{fullD} is the complete set of primitives that are capable of changing all of the joints, creating a high dimensional state-space. We apply motion primitives from MP_{fullD} only to those states s whose end-effector is within d_{fullD} distance from the goal end-effector position. Mathematically, we say that for any state s in the graph: if $dist(ef_{xyz}(s), ef_{xyz}(s_{goal})) > d_{fullD}$, then the set $succs(s) = (\theta_1(s), \theta_2(s), ..., \theta_n(s)) + mp$ for all motion primitives $mp \in MP_{lowD}$, otherwise the set $succs(s) = (\theta_1(s), \theta_2(s), ..., \theta_n(s)) + mp$ for all motion primitives $mp \in MP_{fullD}$. We compute $dist(ef_{xyz}(s), ef_{xyz}(s_{goal}))$ for all states s by running a single 3D Dijkstra's away from the goal end-effector position accounting for obstacles (as described later in the section on heuristics).

The motion primitives we used are multi-resolution as well as multi-dimensional. All $mp \in MP_{lowD}$ are larger motions, allowing the search to reach the goal region quicker. Hence, MP_{lowD} and MP_{fullD} are two different sets, and MP_{fullD} contains smaller motion primitives to allow the search to find a motion to the goal end-effector more precisely. To provide the connections between regions of different resolution in the graph, each joint change for each motion primitive in MP_{lowD} must be of magnitude a that is a multiple of the magnitude by which the joint is changed by motion primitives in MP_{fullD}. Otherwise, the full and low-dimensional regions of the graph may not be connected to each other well enough or even at all.

In our experiments, MP_{lowD} contained eight 4D motion primitives and MP_{fullD} contained fourteen 7D primitives. Figure 2 shows three motion primitives from MP_{lowD}. Each one moves one joint by 8^o.

2) Inverse Kinematics-based Motion Primitives: When a state s is expanded whose end-effector position is within a pre-defined distance to the goal end-effector position, d_{ik}, we use an inverse kinematics solver to generate an additional motion primitive, $mp_{ik}(s, s_{goal})$ for state s. Formally, we state that for any state s with $dist(ef_{xyz}(s), ef_{xyz}(s_{goal})) < d_{ik}$, and the set of successor states, $succs(s) = (succs(s) \cup s_{goal})$ if $mp_{ik}(s, s_{goal})$ exists and is collision free.

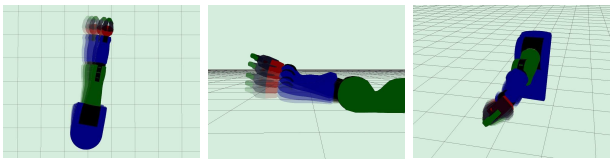

Fig. 2: Three motion primitives from MP_{lowD} are shown here. Each one of these primitives moves one of the joints.

The redundant joint found in most manipulators requires us to feed the IK solver with an initial guess for one of the joint angles. Thus, when computing $mp_{ik}(s, s_{goal})$, we feed IK with one of the angles defined in s. IK then computes an analytical solution for the remaining joints. If the solution does not exist, IK iterates by searching over the entire reachable space for the initially specified joint. It is possible that the IK solution may exhibit divergence, returning a joint configuration that is far away from the seeded configuration. In practice though, the goal region is generally small enough that the solution returned by the solver does not require a large motion to be performed to reach the seed configuration. If IK succeeds, we then construct $mp_{ik}(s, s_{goal})$ as an interpolated path (in the configuration space) from s to the solution returned by IK and also check it for collisions. If it is collision-free, then $mp_{ik}(s, s_{goal})$ is valid and s_{goal} is added to the set of successors of s.

In our experiments, we defined the threshold d_{ik} to be 10cm. Just as with multi-dimensional motion primitives, to compute $dist(ef_{xyz}(s), ef_{xyz}(s_{goal}))$ we use the results of 3D Dijkstra's search originally computed for h_{endeff}. It accounts for obstacles, therefore preventing IK from being called too frequently when the end effector is close to the goal but the direct motion to the goal is blocked.

3) Orientation Solver-based Motion Primitives: When a state s is expanded whose end effector position satisfies the position constraint of the goal, $ef_{xyz}(s_{goal})$, we use an orientation solver to generate an additional motion primitive, mp_{os} for that state. The orientation solver analytically computes the motions necessary to satisfy the orientation constraint, $ef_{rpy}(s_{goal})$ (roll, pitch, yaw angles of the desired end-effector pose), without moving the end effector out of its position, $ef_{xyz}(s)$. The solver computes mp_{os} based on the joint configuration of state s as well as $ef_{rpy}(s_{goal})$. Formally, we state that for any state s, with $ef_{xyz}(s) = ef_{xyz}(s_{goal})$, $succs(s) = succs(s) \cup s_{goal}$ if mp_{os} exists and is collision free.

The orientation solver is based on the premise that the end effector can be reoriented in place, i.e. without displacing the wrist. For example, the orientation solver will work in case of a robot with a ball and socket wrist, because all possible orientations can be achieved by making use of

the joints in the wrist alone. Since an arbitrary orientation is specified in RPY coordinates as (*roll, pitch, yaw*), the output of the orientation solver must consist of increments in three independent joints angles. These incremented changes alter the end effector orientation without displacing the wrist. When the joint space of the robot allows for such reorientation, there are infinite ways in which reorientation can be implemented. The following paragraphs describe how the orientation solver generates the proper motions for the 7DOF arm on the PR2 robot. A very similar strategy can be applied to kinematically comparable robotic arms such as Barrett's WAM.

For the PR2, we restrict the output of the orientation solver to the ordered 3-tuple (*forearm roll, wrist flex, wrist roll*) of motions due to simplicity of expression and the fact that with this convention, the orientation solver can be implemented by an analytical routine. Firstly, consider figure 3, which demonstrates the functionality of the orientation solver. In this example, the robot arm is stretched out straight ahead, and the end effector is in its zero RPY orientation i.e. along the forearm with a roll of zero. Let us say that the desired orientation for the end effector in RPY is $(0, 0, 30^o)$. Since the PR2 wrist cannot yaw, the desired orientation cannot be achieved by purely yawing the wrist. The figure shows how the ordered 3-tuple (*forearm roll, wrist flex, wrist roll*) of motions can be used to attain the desired end effector orientation. Before we proceed to demonstrate the algorithm of the orientation solver, we present a two 3D geometric claims, through which we shall also introduce some notation.

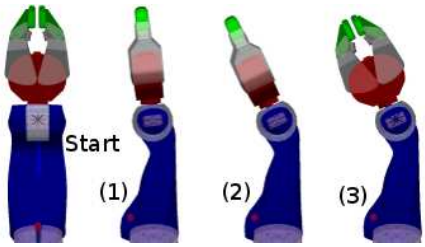

Fig. 3: Orientation solver example, going from RPY $(0, 0, 0)$ to $(0, 0, 30^0)$ l to r; Sequence consists of *forearm roll*$= 90^0$, *wrist flex*$= 30^0$, and *wrist roll*$= 90^0$.

Claim 1 – An arbitrary orientation specified in RPY coordinates (ψ, θ, ϕ) can be represented by two unit vectors v_1 and v_2 as shown in figure 4. v_1 accounts for pitch θ and yaw ϕ, whereas v_2 holds information about roll ψ. The transformation is a straight-forward result and is given by the following equations. This representation is analogous to the axis-angle representation.

$$v_1 = (\cos(\theta)\cos(\phi),\ \cos(\theta)\sin(\phi),\ \sin(\theta))$$
$$v_{2a} = (\sin(\phi),\ -\cos(\phi),\ 0)$$
$$v_2 = R\, v_{2a}, \text{ where}$$

R is the rotation matrix about the axis v_1 and can be obtained by applying the Rodriguez formula.

*Claim 2 – The change in RPY coordinates of the end effector due to an ordered 3-tuple of motions (*forearm roll, wrist flex, wrist roll*) depends upon the orientation of the forearm in the base frame B of the robot*. The claim makes no statement about the exact nature of the dependence, but rather asserts that there is one. To see the reasoning behind this claim, consider figure 5. The forearm and end effector are denoted by their 2-vector representations, with the subscripts F and E respectively. In both the configurations C_1 and C_2 of the arm, the RPY of the end effector is zero in the base frame shown, through C_1 and C_2 differ in their wrist flex. Thus, a change in the RPY coordinates of the end effector does not uniquely map onto the joint space of the arm. However, if the forearm orientation were to be held fixed, then the RPY coordinates of the end effector in the base frame would correspond uniquely to the 3 joint angles forearm roll, wrist flex and wrist roll.

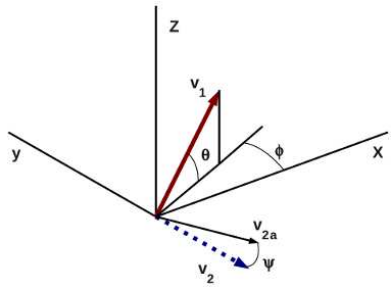

Fig. 4: Converting RPY representation into a 2-vector representation, v_2 shown dotted as it is below the xy plane.

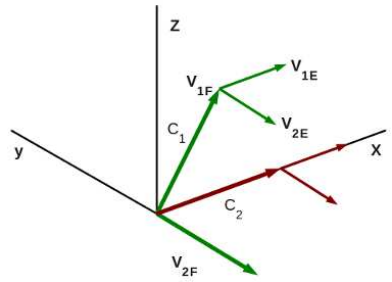

Fig. 5: Same RPY, different arm configurations.

Now, we discuss the algorithm of the orientation solver. Firstly, since the two inputs to the orientation solver are the initial and final orientations of the end effector in RPY coordinates, in the base frame B, it becomes necessary to convert these inputs from frame B to a frame F attached to the forearm, as indicated by the claim 2 above. The frame F is such that the x-axis is along the forearm, and the vector v_{2F} (see figure 5), which corresponds to forearm roll is along the y-axis. This conversion can be implemented through two transformations. 1) Convert end effector RPY in the base frame into the corresponding 2-vector representation in the base frame. Let us say that these vectors are v_{i1B}, v_{i2B}, v_{f1B} and v_{f2B}. Here, the subscript i stands for *initial*, f for *final* and B for *base frame*. 2) Given the rotation matrix R_{FB} of frame F relative to frame B, we convert these four vectors into the corresponding ones v_{i1F}, v_{i2F}, v_{f1F} and v_{f2F}, in

frame F. The ordered 3-tuple of motions can now be found by manipulating these four vectors.

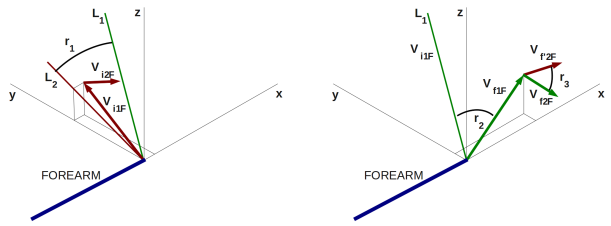

Fig. 6: Explanatory sketch of the orientation solver.

Figure 6 illustrates the functionality of the orientation solver. The forearm and the calculated 2-vectors for the end effector are shown in the reference frame F. Because our desired motion is an ordered 3-tuple (*forearm roll, wrist flex, wrist roll*), the first motion that we need to calculate is the forearm roll (r_1). Since in frame F, a forearm roll causes the projection of a vector on the yz plane to rotate about the x-axis, the magnitude of the forearm roll motion from one orientation to another can be calculated as the angle between the projections of the vectors v_1 corresponding to these orientations. In figure 6, these projections of v_{i1F} and v_{f1F} are denoted by lines L_1 and L_2. The forearm roll is then given as $r_1 = \arccos(u_{L_1} \cdot u_{L_2})$, where u_L is a unit vector along line L, and $p \cdot q$ is the dot product between vectors p and q.

Next, we need to evaluate wrist flex (r_2). Let us denote the rotation matrix corresponding to forearm roll as R_{fr}. The vectors v_{i1F} and v_{i2F} are transformed to $v_{m1F} = R_{fr}\, v_{i1F}$ and $v_{m2F} = R_{fr}\, v_{i2F}$ respectively, after the forearm roll. (These intermediate vectors are not shown in figure 6 to avoid cluttering the diagram.) The wrist flex is then evaluated as $r_2 = \arccos(v_{m1F} \cdot v_{f1F})$. The rotation matrix R_{wf}, which corresponds to wrist flex, is about the axis along the vector $v_{m1F} X v_{f1F}$. This rotation transforms v_{m1F} into v_{f1F}, and v_{m2F} into $v_{f'2F} = R_{wf}\, v_{m2F}$. Thus, after executing the rotations r_1 and r_2, the yaw and the pitch of the end effector will be equal to the final desired values. However, the roll will be different, in general. The wrist roll (r_3) is evaluated as $r_3 = \arccos(v_{f'2F} \cdot v_{f2F})$. The concise algorithm is stated below.

---

1) Compute v_{i1B}, v_{i2B}, v_{f1B} and v_{f2B}
2) Using R_{FB}, compute v_{i1F}, v_{i2F}, v_{f1F} and v_{f2F}
3) Compute u_{L_1} and u_{L_2}
4) $r_1 = \arccos(u_{L_1} \cdot u_{L_2})$
5) Compute R_{fr} as rotation about $(1, 0, 0)$ of angle r_1
6) Compute $v_{m1F} = R_{fr}\, v_{i1F}$ and $v_{m2F} = R_{fr}\, v_{i2F}$
7) $r_2 = \arccos(v_{m1F} \cdot v_{f1F})$
8) Compute R_{wf} as rotation about $v_{m1F} X v_{f1F}$ of angle r_2
9) $v_{f'2F} = R_{wf}\, v_{m2F}$
10) $r_3 = \arccos(v_{f'2F} \cdot v_{f2F})$

The reach of the end effector in RPY space is restricted by joint limits in forearm roll, wrist flex and wrist roll. In the PR2, forearm and wrist roll are continuous but the wrist flex has a finite limit of 126^o on one side and 0^o on the other. Given these limits, there are exactly 2 solutions to the ordered 3-tuple, which differ only in forearm roll. More precisely, forearm rolls in the two 3-tuples are of opposite direction; their absolute values sum to 360^o. In the example in figure 3, the two first sequences are $(90^o, 30^o, 90^o)$ depicted in the figure, and $(-270^0, 30^0, 90^o)$. Our orientation solver is designed to check for both sequences, taking advantage of the fact that even if one sequence will cause a collision, the other sequence may be viable.

B. Cost Function

The cost function is designed to minimize the path length while maximizing the distance between the manipulator and nearby obstacles along the path. Thus the cost of traversing any transition between states s and s' in graph G can 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.

C. Heuristic

The purpose of a heuristic function is to improve the efficiency of the search by guiding it in promising directions. Heuristic-based search algorithms require that the heuristic function is admissible and consistent. In the following sections, we describe the two components of our heuristic function as well as our method for combining them to form a unified admissible heuristic. The first component h_{endeff} has been presented previously [2], whereas the second component h_{elbow} is novel.

1) h_{endeff}: As the ability to plan robustly in cluttered environments is the primary focus of our research, we need a heuristic function that efficiently circumvents obstacles. We use a 3D Dijkstra's search to compute the cost of the least-cost path from the end effector position at a given state to the end effector position at the goal state. Exact details can be found in [2].

h_{endeff} proves to be an informative heuristic in directing the graph search around obstacles in a cluttered workspace. However, h_{endeff} is computed under the assumption that the end effector is a point robot with radius r_{endeff}, the radius of the actual end effector. In doing so, we are treating the arm as an untethered point robot on a 26-connected grid, thus allowing for least-cost paths to be computed that may be infeasible for an end effector limited by the kinematics of the attached manipulator.

Figure 7 displays one such scenario where the starting position of the end effector is below a narrow table and the goal pose is above the table. In the figure it can be seen that the least-cost path from the start configuration leads the end effector around the tabletop to the goal. Considering

the length of the manipulator and its kinematics, such a path is impossible for the end effector to follow. In such a case, h_{endeff} will misguide the search considerably. A more effective search would require a combination of h_{endeff} and some additional kinematic information. The following paragraph explains our method for alleviating this problem.

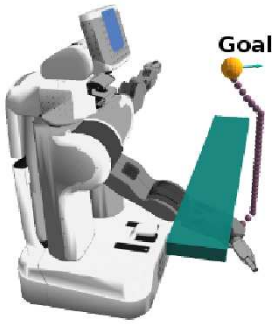

Fig. 7: This is an example when h_{endeff} is capable of guiding the search in a direction that is infeasible for a manipulator to follow. Shown in purple is the least-cost path suggested by h_{endeff} from the start configuration to the goal pose.

2) h_{elbow}: Since the shoulder position is fixed and the lengths of the arm links are known, we can solve for the complete set of possible elbow locations, E_{goal}, such that the end effector satisfies the goal position constraint. The purpose of h_{elbow} is to drive the elbow towards the closest point $e \in E_{goal}$. h_{elbow} is computed in the same way as h_{endeff}, using a 3D Dijkstra's search to compute the cost of the least-cost path from the elbow coordinate, (x, y, z) at state s to the closest $e \in E_{goal}$ while accounting for obstacles in the path.

The following paragraphs discuss the computation of the locus of elbow points for a three link manipulator[1] such as one of the PR2's arms or the Barrett Arm. In short, the locus of points can be solved geometrically by computing the intersection of a circle with a sphere.

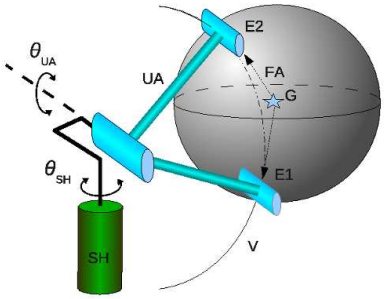

Fig. 8: Given a shoulder pan angle, the upper arm is constrained to move in a vertical plane containing the shoulder link creating a vertical circle with the horizontal axis.

See figure 8, which shows a sketch of the PR2 arm. SH indicates the shoulder link, UA the upper arm and FA the forearm. θ_{SH} represents shoulder pan, and is only a subset

[1]Three links not including the end effector. A link connecting the shoulder pan and shoulder pitch joints, the upper arm and the forearm

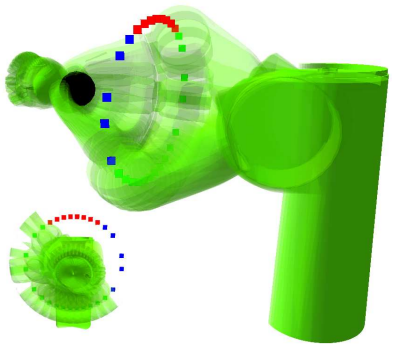

Fig. 9: Elbow locus for end effector goal position constraint (0.60,-0.40,0.8).

of $(0^o, 360^o)$ due to joint limits. The point G represents the end effector goal position, and the sphere S, centered at G with a radius equal to the length of the forearm, represents the locus of all geometrically feasible elbow positions E.

Now, given θ_{SH}, UA is constrained to move in a vertical plane containing SH. The locus of the elbow is then a vertical circle V, with the horizontal axis as shown. Since our aim is to find elbow positions such that the end effector is at G, we need to compute the intersection of V and S. Given SH as shown in figure, this intersection results in two elbow positions, namely $E1$ and $E2$. Note that there are many θ_{SH} such that the corresponding circle V and the sphere S do not intersect. Altogether, a collection of elbow points such as $E1$ and $E2$ for all θ_{SH} within joint limits forms the elbow locus.

In Figure 9, we compute E_{goal} for the end effector goal shown in black. E_{goal} is filtered to eliminate points that violate joint limits, those which put the arm in collision, and those which make it impossible for the goal orientation to be attained while keeping the wrist flex joint angle within its limits. In the figure, blue and red points are elbow positions rejected due to upperarm roll joint limits and shoulder lift joint limits, respectively, whereas green points represent acceptable elbow positions. This analysis can be carried out with other robot arm structures.

3) Combination: Each of these heuristics have strong and sometimes complementary benefits. We combine them by constructing a new heuristic that, for each state s, returns the value $h(s) = max(h_{endeff}(s), h_{elbow}(s))$. Since both $h_{endeff}(s)$ and $h_{elbow}(s)$ are admissible and consistent, the combined heuristic is also admissible and consistent [11]. Experimentation confirmed the utility of the combined $h(s)$ as described above, however it is more efficient to use $h(s) = h_{endeff}(s) + h_{elbow}(s)$. While the summation of the heuristics is not admissible, it is inadmissible by a factor of at most two and can therefore be shown to provide a bound on the suboptimality of the paths returned by a factor of two [11].

In our experimentation, we perform two parallel instances of Dijkstra's algorithm on a 80x70x80 grid prior to every search. One instance computes the distance from the each cell to the end effector goal cell. The other computes the

cost of the path from each cell to the nearest elbow goal cell. The two Dijkstra's searches compute the costs-to-goal for all of the cells in the grid, providing all of the required h_{endeff} and h_{elbow} values. Both the elbow and the end-effector's individual workspaces can be chosen appropriately to maximize efficiency. During testing, we have found that both instances complete in roughly 700ms on the PR2 robot itself.

D. Search

Any standard graph search algorithm can be used to search the constructed graph G. Given the graph's size, however, optimal graph search algorithms such as A* [6] are inapplicable. Instead, we employ an anytime version of A* - Anytime Repairing A* (ARA*) [10].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 in 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.

IV. EXPERIMENTAL RESULTS

To test the capabilities of the motion planner we randomly generated a battery of tests representing different types of realistic manipulation scenarios. We use the 7DOF arm of the PR2 robot as our test platform in these environments. All of the tests require the planner to plan to 6DOF end-effector pose constraints, with a 1cm tolerance around the goal position constraint and an absolute tolerance of 0.05 radians in the roll, pitch, and yaw angles of the end effector. In all of the experiments mentioned in this paper, the planner was given a maximum planning time of two seconds.

See figure 10, which shows the five different types of environments. The tabletop tests are intended to mimic manipulation scenarios. In these tests, the initial arm configuration is randomly generated within a confined region below the right shoulder of the robot. All of the end-effector goal poses are within 16cm of the tabletop, and require roll and pitch angles of zero but differ in the yaw constraint. During the experiments, we varied both the height of the table and the distance of the robot to the edge of the table. The over-under table tests require that the manipulator go from above the table to below and vice versa. The bookshelf tests require that the end effector start on either side of the bookshelf or on a different shelf than the goal pose. The obstacle tests were created by randomly placing cubic obstacles in the workspace of the arm and then randomly generating start configurations and goals. The final set of tests, the cluttered environments, proved to be some of the most difficult, because of the minimal lateral clearance for the elbow. We could not generate a sufficient set of valid cluttered environments to include a table of statistics. From all of the environments we generated, we chose the most difficult and interesting ones and comprised a set of 80 tests we called "Assorted Tests". The cluttered environments were included in the assorted tests and used for the comparison to the sampling-based planner.

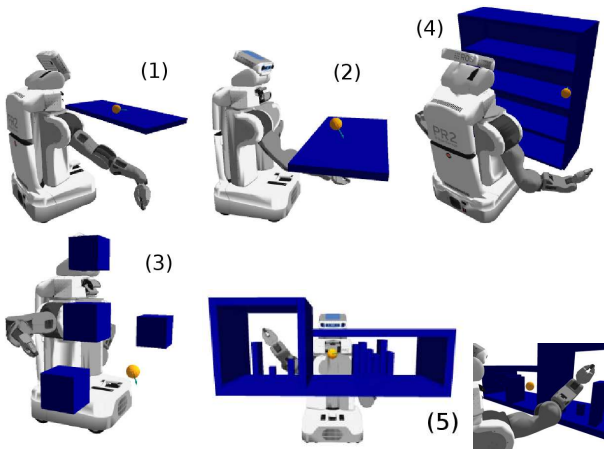

Fig. 10: Shown here are five types of test environments used in our experimental analysis. (1) tabletop manipulation (2) over-under tabletop (3) bookshelf (4) random obstacles (5) cluttered environment

Part of our analysis entailed comparing the effectiveness of using only the orientation solver-based motion primitives, using only inverse kinematics-based motion primitives and combining the two approaches (in addition to the static and multi-resolution motion primitives). Refer to Table I for detailed results. For these tests, we started the planner with an $\epsilon = 100$, setting a maximum planning time of two seconds. The planning times reported in the table do not include the time to compute the heuristics, which was consistently between 0.65 and 0.75 seconds. From these results, we determined that adding inverse kinematics-based motion primitives is better than adding orientation solver-based motion primitives in less cluttered environments, however, the failure rate of using only IK-based motion primitives increases as the environments become more cluttered. The orientation solver-based motion primitives function better in cluttered environments because they generate smaller motions that minimize the risk of collision.

Furthermore, we analysed the effect of switching from the set of coarse 4DOF motion primitives to fine 7DOF motion primitives at different distances from the final goal. Table II shows the effect of this switching distance, d, on the assorted tests. We concluded that although switching farther away from the goal is more costly, it provides a stronger guarantee on completeness. The higher failure rate for a greater switching distance is due to the inability of the planner to finish planning in a timely manner.

Finally, we concluded our analysis by comparing our methods to a sampling-based planner. A cluttered environment was randomly chosen for testing, and we compared the paths returned by multiple calls to a sampling-based planner against those returned by ours. Table III shows comparisons based on the distance metric, which is the ratio of the path

Random Obstacle Tests (128 trials)

solver	cost	expan.	ϵ_{final}	seconds to 1^{st} sol.			failures
				avg	std	max	
OS	33288	4164	10.69	0.114	0.290	1.840	3(2.3%)
IK	21126	2026	4.50	0.042	0.153	1.490	1(0.8%)
Both	20945	2016	4.30	0.042	0.156	1.550	0

Tabletop Manipulation Tests (18 trials)

solver	cost	expan.	ϵ_{final}	seconds to 1^{st} sol.			failures
				avg	std	max	
OS	71000	4798	16.62	0.168	0.218	0.820	2(11.1%)
IK	57875	3548	11.50	0.303	0.553	2.020	2(11.1%)
Both	67333	3463	13.78	0.306	0.495	1.850	0

Table Over-Under Tests(38 trials)

solver	cost	expan.	ϵ_{final}	seconds to 1^{st} sol.			failures
				avg	std	max	
OS	72194	4419	14.94	0.334	0.362	1.900	1(2.6%)
IK	52731	3493	10.61	0.307	0.431	2.000	12(31.6%)
Both	62368	3408	10.26	0.267	0.207	0.750	0

Bookshelf Tests(8 trials)

solver	cost	expan.	ϵ_{final}	seconds to 1^{st} sol.			failures
				avg	std	max	
OS	75250	4965	16.00	0.086	0.050	0.200	1(2.6%)
IK	41143	3486	6.86	0.039	0.012	0.060	12(31.6%)
Both	42750	3619	7.50	0.059	0.031	0.110	0

TABLE I: Summary of experimental results.

length to the Euclidean distance from the start position of the end effector to the goal pose. While our search-based planner is deterministic and always gives the same score, the scores of the sampling-based planner vary drastically. We believe that such variability can be counter-productive in situations where repeatability is a requirement.

Assorted tests (80 trials)

d	cost	expan.	ϵ_{final}	failures
$20cm$	68016	2760	14.19	18(22.5%)
$40cm$	89128	2865	17.83	33(41.2%)

TABLE II: The effect of changing distance d, the threshold for switching between MP_{lowD} and MP_{highD}, on the efficacy of the planner.

Cluttered test (search-based planner score: 4.46)

avg	std	max	min
6.43	3.29	13.23	1.65

TABLE III: Showing variability of paths returned by sampling based planners, based on the distance metric. The test used was a randomly chosen highly cluttered environment.

V. CONCLUSION

In this paper we have presented a search-based motion planning algorithm for manipulation that is capable of planning efficiently for manipulators with many degrees of freedom. In our approach, we introduced adaptive motion primitives that plan efficiently and precisely using both pre-defined multi-dimensional actions and primitives that are generated on the fly by analytical solvers. The algorithm relies on an anytime graph search to generate solutions quickly, as well as provide theoretical guarantees on the completeness, consistency and provides a bounds on the sub-optimality of the solution cost, under state space and action space constraints. The search is facilitated by a combination of two heuristics that aid in coping with obstacles in the environment. While the algorithm was tested on the PR2, it is general enough to apply to other robots with kinematically similar arm structures. Our experimental analysis shows that the use of adaptive motion primitives in search-based planning for manipulation can lead to very efficient planning. In particular, these adaptive motion primitives significantly improve search performance, while maintaining the same theoretical guarantees as our planner that used static motion primitives.

VI. 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.

REFERENCES

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

[2] B. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[3] 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.

[4] E. Frazzoli and S. Karaman. Incremental sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 2010.

[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] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

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

[14] 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.