Single- and Dual-Arm Motion Planning with Heuristic Search

Benjamin Cohen Sachin Chitta Maxim Likhachev

Abstract— **Heuristic searches such as A* search are a popular means of finding least-cost plans due to their generality, strong theoretical guarantees on completeness and optimality, simplicity in implementation and consistent behavior. 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 approach to motion planning for manipulation that does deal effectively with the high-dimensionality of the problem. Our approach achieves the necessary efficiency by exploiting the following three key principles: (a) representation of the planning problem with what we call a *manipulation lattice graph*; (b) use of ARA* search which is an anytime heuristic search with provable bounds on solution sub-optimality; and (c) use of informative yet fast-to-compute heuristics. The paper presents the approach together with its theoretical properties and shows how to apply it to single-arm and dual-arm motion planning with upright constraints on a PR2 robot operating in non-trivial cluttered spaces. An extensive experimental analysis in both simulation and on a physical PR2 shows that, in terms of runtime, our approach is on par with other most common sampling-based approaches despite the high-dimensionality of the problems. In addition, the experimental analysis shows that due to its deterministic cost-minimization, the approach generates motions that are of good quality and are consistent, i.e. the resulting plans tend to be similar for similar tasks. 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 controlling or interacting with the robot.**

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 [11] 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 [19]. Second, the generality of heuristic searches allows one to incorporate complex cost functions and complex constraints and to represent easily arbitrarily shaped obstacles with grid-like data structures [8], [17]. Finally, heuristic searches provide good cost minimization and consistency in the solutions. Consequently, heuristic search-based planning has been used successfully to solve a wide variety of planning problems in robotics.

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

Despite the wide popularity of heuristic searches, they typically have not been used for motion planning for high-DOF robotic manipulators. 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, we use a *manipulation lattice graph* to represent the planning problem. A manipulation lattice graph is a sparse representation in which the states correspond to the configuration of the robot while the edges represent short motion primitives. Any path in the graph is kinematically feasible for the robot. This representation was designed to specifically handle the complexities of manipulation through the use of *static* and *adaptive* motion primitives and by decoupling the problem when appropriate by varying the dimensionality of the lattice. Second, 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* [18], that finds solutions with provable bounds on suboptimality and improves these solutions until allotted time for planning expires. Third, the solutions found in a low-dimensional simplification of the workspace can serve as highly informative heuristics and can therefore efficiently guide the search despite the kinematic constraints that are inherent to manipulation.

This paper presents our approach and then explains how to apply it to single-arm and dual-arm planning while effectively handling the complex constraints that arise in each domain. We motivate and describe in detail the different components of the manipulation lattice graph and the heuristics used for each problem. Our experimental results help to demonstrate the effective cost minimization and competitive planning times for our approach . We also show, through an additional set of experiments and multiple trajectory similarity measures, that in addition to being deterministic, our approach generates *consistent* solutions. Consistency here implies that given similar input, our approach produces similar output. Consistency helps in making the robot's motions more predictable. Predictability in a robot's motions is required if a human is expected to operate it safely or be safe while working in close range of it.

The paper is organized as follows. In Section II, we first briefly discuss other types of approaches to motion planning. We present an overview of heuristic search-based planning for manipulation in Section III. We describe how our approach is applied to single-arm and dual-arm planning in Section IV and V. Next, we present extensions to our planners that naturally arise from the fact that we employ a heuristic search-based approach. Section VII and Section VIII present the results of experiments performed on the PR2 robot both in simulation and on the actual robot. Our results include comparisons to sampling-based approaches in terms of solution quality, performance and consistency of the resulting paths.

II. RELATED WORK

Sampling-based motion planners [15], [16], [2] 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 them. Sampling-based methods have also been extended to support motion constraints through rejection sampling [23].

Our approach to motion planning differs from these algorithms in several aspects. First, sampling-based motion planners 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 shortcutting [12] and 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, heuristic search-based planning tries to find 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 an objective function is important and when consistent behavior is expected.

A recently proposed incremental sampling based planner based on RRT* [9] additionally provides guarantees on solution optimality and completeness. This is an important step in the field of sampling-based motion planning towards a good cost minimization. However, as our experimental analysis confirms, the random nature of sampling-based planners including RRT* makes it difficult to generate good quality solutions fast and to generate consistent motions across similar runs.

Several motion planning algorithms have also been developed that can optimize cost functions such as obstacle costs or path smoothness [13], [22], [14]. CHOMP [22] 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. However like sampling-based planning, CHOMP relies on randomization via random restarts throughout the algorithm to help deal with environmental complexities. STOMP [14] offers the ability to optimize general cost functions and has been shown to work well for manipulation problems. Similarly, our approach offers the ability to minimize more general cost functions and, in addition, provides bounds on global solution suboptimality with respect to the constructed graph.

Our approach in this paper builds on earlier work in [4], [5], [6] where we presented approaches to particular problems involving motion planning for single-arm and dual-arm manipulation tasks. In this work, we present a more general unified search-based planning framework for manipulation which encompasses all the earlier approaches. The application to single-arm and dual-arm planning can thus be considered as specific instances of our general approach. We also present more informative statistics based on a series of comparative tests that attempt to quantify the performance of our planning vis-a-vis other planning approaches.

III. HEURISTIC SEARCH-BASED PLANNING FOR MANIPULATION

Heuristic search-based planning has four major components - graph construction, an informative heuristic, the cost function and the actual search itself. The design of these core components directly determines several important factors including the planning time, memory footprint, smoothness of the trajectory, and whether or not a motion that the planner generates is kinematically and dynamically feasible for the robot to execute. In the example of motion planning for manipulation, we define the goal of the graph search itself as the search for the least cost path in the constructed graph from a state that corresponds to the initial configuration of the manipulator to a state for which the pose of the end-effector satisfies a *goal constraint* in work space. Here the goal constraint essentially determines a region in space within which we want the end-effector of the robot to lie once it completes its planned motion. In Section VI, we will show that this notion of a goal can be extended easily to incorporate more general constraints.

Despite the wide popularity of heuristic searches in the path planning community, especially for navigation, they have yet to be seriously considered for motion planning for manipulation. The obvious reason why heuristic searches have not extended from navigation to manipulation is the high-dimensionality of the motion planning problem for manipulation. While the high dimensional state-space is a difficult problem in its own right, we have also observed a set of nuances specific to manipulation that greatly add to the complexity of the search. Here are three of them:

1) While the goal constraint is defined in the workspace of the end-effector, the feasibility check for a given state is performed in joint space.
2) The goal constraint is comprised of positional and rotational elements, which at times during the search can be perceived as two different problems entirely.
3) For successful grasping and manipulation to occur, minimal to zero tolerance around the goal constraint

may be allowed, suggesting the need for a fine discretization of the state space.

To overcome these issues, we rely on three critical features of our approach that together make the application of search-based planning to motion planning for manipulation more tractable: (1) A manipulation lattice graph representation, (2) Anytime search algorithms and (3) Informative heuristics. We will now describe these three features in more detail in addition to describing the core features of our algorithm itself.

A. Manipulation Lattice Graph Representation

The first novel feature of our approach is to employ a non-uniform resolution lattice search space with non-uniform dimensionality whose edges correspond to a predefined set of actions, *static motion primitives*, as well as *adaptive motion primitives*, or actions that are generated at runtime. We call this novel approach, a *manipulation lattice graph*. Unlike a standard lattice graph representation, our approach is capable of decoupling the search space when appropriate and permits the use of a coarser discretization without sacrificing the ability to satisfy an arbitrary goal constraint. Our approach remains true to standard lattices, in that it is a sparse representation where every path in the lattice represents a kinematically feasible motion.

The manipulation lattice graph, was inspired by the success of lattice-based planners in planning dynamically feasible trajectories for navigation [17]. A 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 [20]. 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 of the joints in the manipulator we are planning for. The transitions in E correspond to two types of short, *atomic*, 100 ms duration actions we call motion primitives[1], *static* and *adaptive*. We define a state s as an n-tuple $(\theta_0, \theta_1, \theta_2, ..., \theta_n)$ for a manipulator with n joints. It is important to note that the graph is constructed dynamically 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 positions of neighboring states.

[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.

We refer to the pre-defined set of actions that can be performed at any given state, as *static motion primitives*. These actions are chosen before the search begins and their purpose is to uniformly explore the space for a valid path. Given that this set is the majority of all motion primitives used, it has a major impact on the branching factor of the graph search. When designing the set of static motion primitives, a fine balance must be found between the speed of the search and the density of the exploration. We briefly discuss selecting motion primitives further in Section IX.

We will now present the techniques used by the manipulation lattice to combat the high dimensional statespace and the complications specific to manipulation.

Adaptive Motion Primitives. Many problems in manipulation may require the robot's end effector to achieve a very specific goal configuration, e.g. if the goal of a motion is to ultimately grasp an object. Such precise positioning may be difficult to achieve on a lattice that is derived with discretization. A fine discretization will make the search more computationally expensive but a coarse discretization may make it difficult to achieve any arbitrary goal constraint. To that end, we limit the effects of discretization through the use of continuous solvers that compute the edges to connect any given state to the goal state. We call these motions a type of *adaptive* motion primitives, or primitives that are dynamically generated on-the-fly. Adaptive motion primitives are actions that don't belong to the static set and that are created as needed during a state expansion, given that the state meets a certain set of pre-defined criteria. In the motivating example of an adaptive motion to reach the goal, we say that when the state, s, that is being expanded represents an end effector position that is close to the goal, a motion primitive is generated connecting s to s_{goal}. By *snapping* to the goal pose, the search is shortened and it is capable of satisfying any arbitrary goal constraint in spite of the discretization of the state space. It is important that the adaptive motions are used in conjunction with the static set so that a systematic search is still being performed. It is obvious that *adaptive* motion primitives play a major role in compacting the graph for a high dimensional domain such as manipulation. Another appropriate name for these motions would be *runtime* motion primitives.

One example of a continuous solver-based adaptive motion primitive, or *amp*, that we use is *inverse kinematics-based*. When a state s is expanded whose end-effector position, $ef_{xyz}(s)$, is within a pre-defined distance to the goal end-effector position, d_{ik}, we use an inverse kinematics (IK) solver to generate an additional motion primitive, $amp_{ik}(s, s_{goal})$ for state s. Let $succs(s)$ denote the set of successors in the graph for a state s. Formally we state, that for any state s with $dist(ef_{xyz}(s), ef_{xyz}(s_{goal})) < d_{ik}$, $succs(s)$ = $(succs(s) \cup s_{goal})$ if $amp_{ik}(s, s_{goal})$ exists and is collision free. If IK succeeds, we then construct

$amp_{ik}(s, s_{goal})$ 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 $amp_{ik}(s, s_{goal})$ is valid and s_{goal} is added to the set of successors of s.

Non-uniform Dimensionality. Planning in a high dimensional lattice is computationally expensive and can require 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 may be 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 in computing a feasible path to the vicinity of the desired end-effector pose. Once it does get close to the desired end-effector pose, changing additional joints may become necessary in order to satisfy orientation constraints and to maneuver the end-effector in cluttered spaces.

This observation motivated us to generate a set of static motion primitives that varies in its dimensionality. A subset of this full set of motion primitives can be used to quickly search for a path to the goal region. These motion primitives are chosen such as to result in 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 that can change only 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 in the subset).

MP_{fullD} is the complete set of primitives that are capable of changing all of the joints, creating a full 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 breadth first search starting from $ef_{xyz}(s_{goal})$ (as described later in the section on heuristics), thus accounting for obstacles in the environment as well.

Non-uniform Resolution. 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 get to the general goal region quicker. MP_{fullD} contains shorter motion primitives to allow the search to find a motion to the goal more precisely. Thus, MP_{lowD} and MP_{fullD} are two different sets. To provide the connections between regions of different resolutions in the graph, it is important that each joint change for each motion primitive in MP_{lowD} is of a magnitude 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 well connected. If there are no states in the $lowD$ portion of the graph that coincide with $fullD$ states, then the two regions of the graph may not be connected at all. Even if they are connected, there may not be a path from start to goal between them. Thus, setting the graph resolutions to be multiples of each other ensures that there are plenty of paths to choose from.

B. Anytime 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* [11] are infeasible to use. 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* [18], that quickly finds an initial and possibly sub-optimal solution and repairs it while deliberation time allows, efficiently reusing its previous efforts. Despite its *anytime* capabilities and its adept reuse of previous work, the efficiency of each particular iteration is heavily dependent on the quality of the heuristic guiding it. Also, the graph representation used to characterize the planning problem is a major factor in determining how many expansions are necessary to reach the goal state. The algorithm guarantees completeness for a given graph G and provides a bound ϵ on the sub-optimality of the solution at any point of time during the search. In the context of our problem, the algorithmic guarantees on completeness and optimality are w.r.t. to the set of motion primitives used and the resolution of the configuration space.

ARA* is capable of finding a solution of bounded sub-optimality quicker than finding the optimal one by multiplying the heuristic by an inflation factor $\epsilon > 1$. The cost of the solution generated will be at most ϵ times the cost of the optimal solution. If the heuristic is a good approximation of the problem we are solving then it will serve as an informative guide, finding the solution quicker by expanding fewer states. If the inflation factor ϵ is large and the heuristic is very informative then a solution can be found rather quickly. However, if the ϵ is large and the heuristic is a poor approximation of the problem, then misplaced trust has been put in the heuristic, allowing it to guide the search into deep local minima. In Figure 1, the left image shows the area searched by a standard A* search ($\epsilon = 1$) in yellow from the start on the left to the goal on the right. In the image on the right, the informative heuristic is weighted by some $\epsilon > 1$ causing the search to rush towards the goal with fewer state expansions (smaller yellow area).

If a path is found using the initial epsilon within the time allotted, then a follow up 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. ARA* gains an additional efficiency by not re-computing

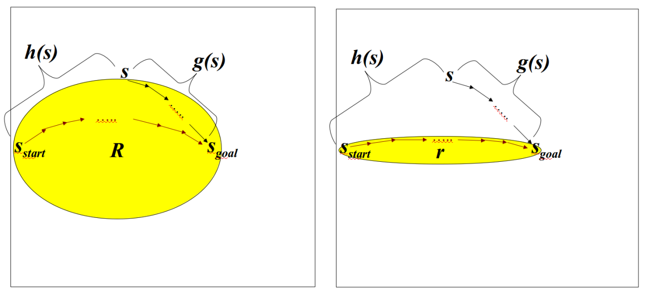
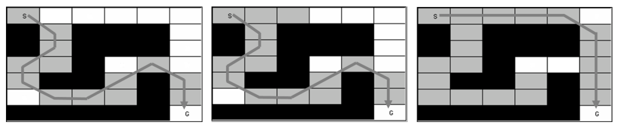


Fig. 1: An A* search is performed on the left ($\epsilon = 1$). The area in yellow represents all of the states that were expanded to find the optimal path. In the A* search with inflated heuristics shown on the right, fewer expansions were required to reach the goal.

the states that it already computed in its previous search iterations. Figure 2 contains a series of weighted A* searches with decreasing ϵ from left to right. It requires a total of 48 states to be expanded to find the optimal path. For the same problem, ARA* is used in Figure 3, and a total of 23 state expansions across all runs to find the optimal solution.

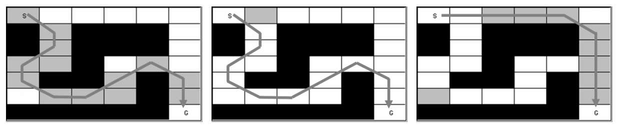


(a) $\epsilon = 2.5$
13 expansions

(b) $\epsilon = 1.5$
15 expansions

(c) $\epsilon = 1.0$
20 expansions

Fig. 2: Weighted A*: Optimal solution is found after 48 states are expanded.



(a) $\epsilon = 2.5$
13 expansions

(b) $\epsilon = 1.5$
1 expansion

(c) $\epsilon = 1.0$
9 expansions

Fig. 3: ARA*: Optimal solution is found after only 23 state expansions because previous efforts are reused.

As in any heuristic search, any arbitrary cost function can be used here. In manipulation it is conceivable that one might want to minimize the path length in joint space, the path length of the end effector, the power consumed by the arm, or a combination of any number of these. The cost function we used is designed to minimize the path length 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.

C. Informative Heuristics

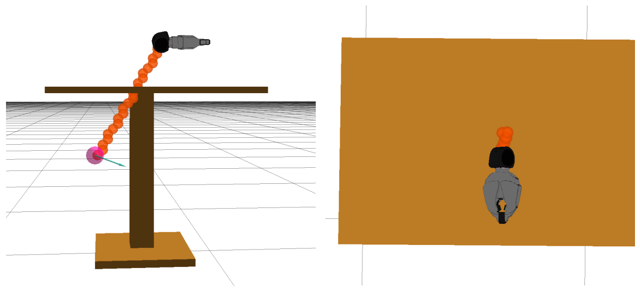
For a heuristic function to be most informative, it must capture the key complexities associated with the overall search, such as mechanism constraints or the environment complexities. Given the high degree of freedom in motion planning for manipulation, it is even more imperative that the heuristic has shallow local minima to prevent the search from getting stuck.

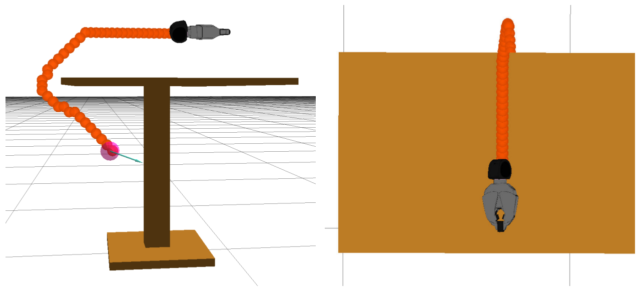
A common approach for constructing a heuristic is to use the results from a simplified search problem (e.g. from a lower-dimensional search problem where some of the original constraints have been relaxed). Heuristic-based search algorithms require that the heuristic function, h, is admissible and consistent. 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 the heuristic at 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'.

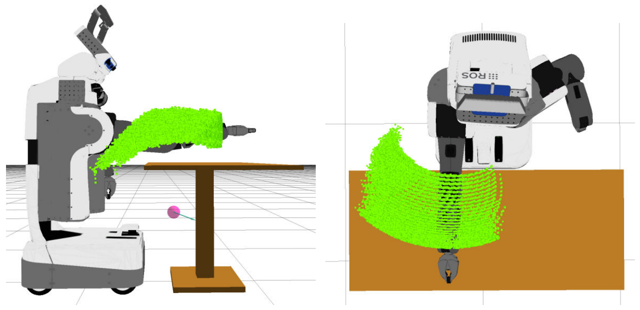
A common task in manipulation is to move the end-effector to a desired goal position. Thus it is useful to employ a heuristic function that is informative about the end-effector position and orientation, (x, y, z, r, p, y). We use a single heuristic that guides the search towards achieving the (x, y, z) component of the goal constraint. In our earlier work [5] [6], we have also explored using different heuristics to guide the search, one guiding towards the position of the goal constraint and one guiding towards the orientation of the goal constraint. However, combining multiple heuristics effectively is still an open problem for future research.

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 some dimensionality is a standard technique in creating an informative heuristic function and we use a 3D breadth first search (BFS) to compute the costs of the least-cost paths from every cell in a grid to the cell that corresponds to the goal position (x, y, z) while avoiding obstacles. During the planning, the heuristic component $h(s)$ 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 breadth first search for the cell with those coordinates. This heuristic proves to be an informative heuristic in directing the graph search around obstacles in very cluttered workspace.

In Figure 4 we illustrate the effectiveness of the 3D breadth first search heuristic through a comparison to euclidean distance as a heuristic. The robot used in this illustration is the PR2 robot (see Section IV for more details about this robot). In Figure 4a and Figure 4b, the least cost path is drawn from the initial pose of the end effector to the goal pose beneath the table, represented by the pink sphere and


(a) The least cost path according to $h_{Euclidean}$.


(b) The least cost path according to h_{BFS}.


(c) Using $h_{Euclidean}$ inflated by $\epsilon = 100$, a solution was found after 35,333 expansions.

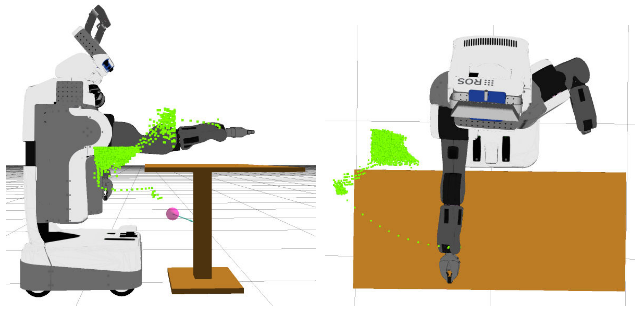

(d) Using h_{BFS} inflated by $\epsilon = 100$, a solution was found after 2,100 expansions.

Fig. 4: In this example, the pink sphere with cyan arrow represents the goal pose for the right end effector. The path shown in red, is the shortest path suggested by the heuristic for the end-effector at its initial position. The green cubes represent $ef_{xyz}(s)$ of each expanded state when the respective heuristic function is used.

cyan arrow. While h_{BFS} suggests a path around the edge of the table, $h_{Euclidean}$ offers an infeasible path through the tabletop. We performed an experiment in which we used each of these heuristics to plan a path for the arm of the PR2 from above the table to the goal below. In the experiment we inflated the heuristic with an $\epsilon = 100$ to highlight how effective it is. In Figures 4c and 4d we use green cubes to represent the $ef_{xyz}(s)$ of all of the expansions required to reach the goal. Figure 4c demonstrates an inefficient and almost uniform search within the workspace of the arm, requiring 35,333 expansions to find a path with $h_{Euclidean}$. While, Figure 4d shows that with h_{BFS}, a more direct route is taken and only 2,100 states are expanded to compute a solution.

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 BFS may not be reachable by the manipulator. In these situations, the heuristic becomes detrimental to the efficiency of the search. An example of such a scenario is if the heuristic directs the search outside of the reachable workspace of the arm. We found that this rarely happens in practice. Nevertheless, we discuss techniques to avoid such cases in Section IX.

For efficiency, the heuristic for a given state is computed only as needed. Upon request, if the desired state has not been reached by the BFS tree, the breadth first search is expanded out from the goal state until the desired state is found. This way we avoid needlessly performing the BFS over the entire grid apriori.

An interesting additional benefit of the adaptive motion primitives presented earlier is that they can assist the search in reaching the orientation constraint of the goal pose without the guidance of a heuristic. This is especially important in a problem such as manipulation which requires a goal pose for the end effector that has two components - a position and orientation constraint. It is difficult to develop an informative heuristic in such situations that can effectively guide the search towards both components at the same time. Adaptive motion primitives, especially the primitive that snaps to a goal pose, are extremely useful in such situations.

IV. SINGLE-ARM PLANNING

The approach that we described in section III is capable of planning to a goal constraint defined as a joint configuration for the manipulator or as a pose constraint for the end-effector of the manipulator. We chose to design our motion planner to plan directly to an end-effector pose instead of requiring it to plan to a joint configuration. This allows the planner to converge to a joint configuration of its choice based on the particular cost function being used. Such a choice can have several benefits:

1) In a typical pick and place application, after an object is detected and localized, a grasp planner is used to generate an end effector pose capable of grasping the object. The end effector pose it generates is a logical goal input to the manipulation planner.
2) After a desired end effector pose is determined, it can be time consuming to search through the possible

inverse kinematic solutions for one that is collision free. Also, it is possible that the joint configuration solution itself may be safe, but no collision free path from the initial configuration exists.

3) If a valid joint configuration is not found but a tolerance on the goal constraint is allowed, then some sort of search through the allowed goal region would have to be performed so that a valid joint configuration can be computed before planning can begin. It is not straight forward how to search through the allowed goal region for a valid configuration in a principled and efficient way. These problems are avoided when planning for the end-effector to any pose in the desired goal region.

We will now present the particular nuances that allow us to adapt our generic approach presented in Section III to the problem of motion planning for a single arm.

A. *Manipulation Lattice Graph*

The representation that we described in Section III can be directly applied to the single-arm planning problem. We construct an n dimensional statespace when planning for an arm with n joints. Each state is represented by an n element vector whose elements correlate to actual joint positions. Each motion primitive is an n element vector of velocities. Figure 5 shows two views of a motion primitive being performed by one of the 7 degree of freedom arms of the PR2.

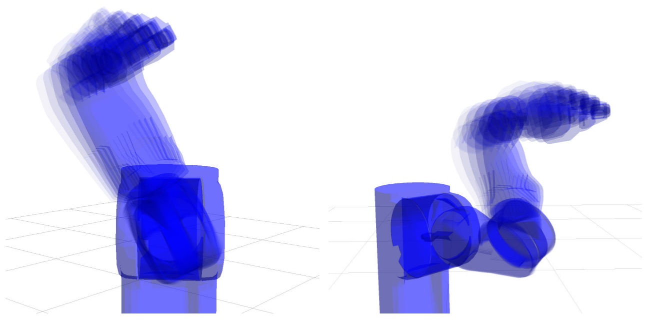


Fig. 5: Two perspectives of a static motion primitive in which the upper arm is rolled $11.5°$, the elbow is flexed $8°$ and the wrist is pitched $-8°$, it is represented by $(0, 0, 11.5°, 8°, 0, -8°, 0)$.

In addition to the adaptive motion primitive, $amp_{ik}(s, s_{goal})$ that we described earlier, for single-arm planning we use another continuous solver-based amp, which we call an *orientation solver-based* primitive, or $amp_{os}(s, s_{goal})$. 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, amp_{os} for that state. The orientation solver computes the proper 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 amp_{os} exists and is collision free. The orientation solver is based on the premise that in many cases, the end-effector can be reoriented in place without displacing the wrist. Thus, 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.

B. *Heuristic*

To make the heuristic we presented in section III more representative of the actual search, we represent the end-effector using its inner sphere, i.e. the largest sphere that is contained completely within the volume of the end-effector. In our implementation, this implies that we are effectively adding an extra padding to the obstacles equal to the radius of this sphere when running the 3D breadth first search to compute heuristics.

V. DUAL-ARM PLANNING

Dual-arm manipulation is an important skill for a robot to have given that many of the objects we interact with on a day to day basis are too heavy to be lifted by a single arm or are too large to grasp with one. Many dual-arm tasks come with a natural requirement that the object be kept upright throughout the entire path, such as carrying a tray with food or drink on it. In this section, we apply the framework presented in Section III to motion planning for dual-arm manipulation with an upright constraint.

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. We now discuss slight modifications to the general design proposed in Section III.

A. *Manipulation Lattice Graph*

In Section IV we presented our method of planning for single-arm manipulation in which we represented the configuration space of the arm in joint space. Thus, if we needed to plan motions for a robot arm with seven joints, it would result in a graph with seven dimensions. The combination of informative heuristics, anytime graph search and adaptive motion primitives addressed the high dimensionality of the state space. However, if we were to construct a graph in the same way for dual-arm planning then we would end up with a 14 dimensional state space and the three key components would become less effective. Fortunately though, we can exploit the natural dimensionality reduction that stems from the two constraints we mentioned above - the constraint arising from the dual-arm grasp on the object and the upright orientation constraint. We will illustrate our approach for the dual-arm PR2 robot where each arm has 7 degrees of freedom.

Given the global pose of the object and the positions of one degree of freedom in each arm, we can compute the complete configuration of each arm. That is, there is a one to one mapping between the 14 dimensional joint space of the two arms and the 8 dimensional space represented by the object pose and two free angles (one for each 7 degree of freedom arm), represented as $(x, y, z, roll, pitch, yaw, \theta_1, \theta_2)$. When an upright orientation constraint is considered, the representation can be reduced further to 6 dimensions, $(x, y, z, yaw, \theta_1, \theta_2)$, because the roll and pitch of the object are fixed at zero throughout the path. This representation enables us to plan by constructing the graph in 6 dimensions. The 6 dimensional states can be mapped back to the full 14 dimensional space whenever required, e.g. for collision checking.

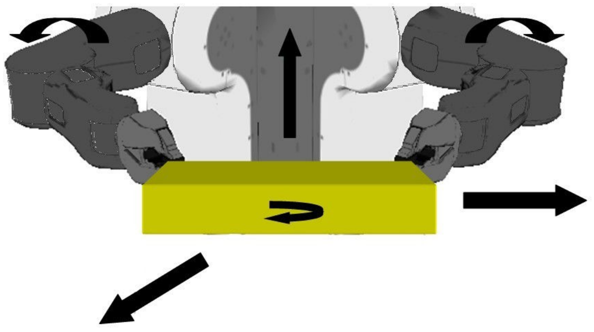


Fig. 6: The six degrees of freedom in the statespace.

Similar to our approach for the single arm, we use a manipulation lattice graph here as well. The states in S are the set of possible (discretized) 4 degree of freedom 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 6 for a visualization of the 6-tuple for the PR2 robot. Note that as mentioned in Section IV, we dynamically construct the graph during the graph search because pre-allocation is infeasible for the 6 dimensional graph. In this domain we define a motion primitive as a vector of translational and rotational velocities of the object in the global frame combined with the velocities of the two redundant joints. Each motion primitive again has a duration of 100 ms.

Before a successor of state s, s' can be added to the graph it must be checked for feasibility, i.e., 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. If a solution is found for each arm, then we can check if the state is valid, i.e. forward simulate the motion of both arms corresponding to linear interpolation between configuration in s and the solutions returned by IK and then check for collisions.

During the expansion of a given state s, it is not uncommon for the inverse kinematics solver to fail when computing a solution for one or both of the arms when determining the feasibility of a potential successor, state s'. If that occurs, then rather than just reject the invalid successor completely, we search over the redundant joint space for that arm for a valid joint configuration that satisfies the (x, y, z, yaw) component of s'. If a solution exists, then we generate an *adaptive motion primitive* that is essentially defined as the action that was used to reach the invalid state s' except it has a new value for θ_1, θ_2 or both. This results in a successor state, s'', whose coordinates represent the same object pose as the coordinates of s' but with a possibly new value for θ_1, θ_2 or both.

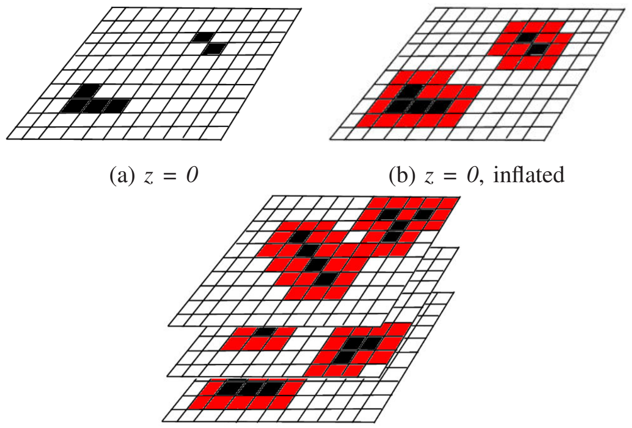
B. Heuristic

In the heuristic we used for single-arm planning, we represented the end effector as its inner sphere during the computation of the BFS. The same heuristic can be used here as well but given the upright constraint, we can modify it to make it more informative. Instead of modeling the object as a sphere when performing the 3D breadth first search, we instead model it as a cylinder, or a stack of cylindrical discs, because we are constraining the object from rolling or pitching. The radius of the cylinder at a given z height, is the radius of the inner circle of the object at height z, i.e. the circle centered at the geometric center of the object (in the *xy* plane) at height z, with the largest radius such that it is completely contained within the object footprint.

To compute the heuristic, we iterate through n height levels of the object and for each, we create an inflated xy plane of the map. 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+*h_{object}*)* are collision free. A detailed example can be found in Figure 7.

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



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

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

Fig. 7: The obstacles are shown in black and the inflated cells are red. The radius of the inner circle of the object is 1 cell and h_{object}=3. Upon a lookup for *h(x,y,z)*, *h(x,y,z)*, *h(x,y,z+1)* and *h(x,y,z+2)* must be checked to determine if *h(x,y,z)* is a valid state.

the planner, i.e., the planner may not find a solution even if one exists. More on this can be found in Section IX.

VI. EXTENSIONS

The generic nature of heuristic-searches make them easily extendable in many ways. We now present four capabilities of our approach that are straight forward to implement and that come at little or no cost in its performance.

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 the upright constraint required to manipulate a tray with glasses on it, as discussed in Section V. Our approach supports arbitrary path constraints expressed as bounds on the position or orientation of a certain link as a required joint position of a specific joint. In addition to simply rejecting invalid successors, path constraints can be implemented as adaptive motion primitives that adjust previously rejected static motion primitives to satisfy the constraint. It is very common that path constraints can provide a decent speedup to the planner by effectively shrinking the statespace.

Goal Sets. Another feature of our approach is the ability to handle multiple goal poses as input and return a path to the goal pose with the minimal path cost overall. This can prove useful when a grasp planner finds multiple feasible grasp poses for manipulating an object and heuristics to choose between them are not obvious. In this case, the motion planner will compute a path to the grasping pose with the least-cost path.

Goal Regions. Many motion planning problems do not require the end effector to achieve an exact pose or one within a goal set. Rather, the problem may call for the end effector to be placed anywhere in some particular goal region to perform a task. The region can be defined for the end effector in the space of its position, orientation or both. Like all heuristic searches, our approach is capable of planning to goal regions without any algorithmic modifications. An example of using sampling based approaches to plan to a goal region can be found in [1].

Invalid Goal States. When planning for manipulation it would be preferable if the motion planner was capable of planning to the actual grasp pose determined by the grasp planner. Since most planning approaches are incapable of planning to an invalid goal state (e.g. a state in collision), the alternative method is to translate the grasp pose away from the object and then plan to the translated pose. To pick up the object after the planned path is executed, an undesirable open loop motion would be required to move the end effector the translated distance. However, for heuristic searches, planning to an invalid goal state is a viable option. With our approach, one can determine a set of neighboring states that are valid before planning. Then the set can be used as a goal set for the search. Since, goal sets have no negative effect on the efficiency of the search, the only additional computation stems from computing the *virtual goal set*.

VII. SINGLE-ARM EXPERIMENTAL RESULTS

A. Experimental Setup

To measure the performance of our planner and compare it against other approaches, we carried out a set of benchmark experiments in different simulated environments. The set of planners we compared our approach to are RRT* [9] and RRT-Connect [16]. The cost function for RRT* is the distance traveled in joint space. Unless stated otherwise, RRT* was configured to return the first solution that was found. All these planners are implemented in the OMPL library[7]. More information on the benchmarking infrastructure used for our experiments, as well as details on the environments, can be found in [3].

All experiments were carried out using a simulated model of the PR2 robot. The PR2 robot is a two-armed mobile manipulation platform with an omni-directional base. It carries a suite of onboard sensors that are useful for modeling the environment in 3D. This includes a tilting laser range finder, a stereo camera and RGB-D sensors. The sensors generate 3D point clouds representing the environment that can be incorporated into the collision environment used by the planners. The PR2 also has sufficient onboard computing to carry out fast motion planning in addition to any realtime 3D processing required to build an efficient representation of the environment.

The environments we used are shown in Figure 9. In each environment, multiple goal locations are defined for the end-effector of a robot. The planners were instructed to plan between successive goal positions for the arm of the robot. Each experiment, between a start and goal position, was carried out multiple times to ensure that a total of at least 30 experiments were run in each environment for each planner. Details of the planning requests can also be found in [3]. Each planner was given a maximum of 60 seconds to

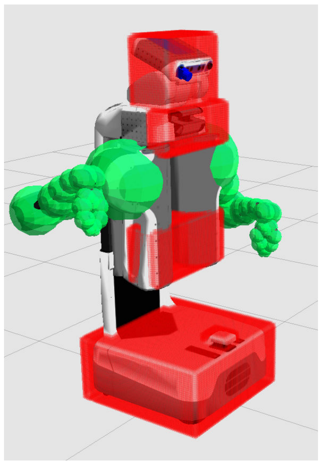


Fig. 8: The simplified collision model used in our experiments. The green spheres represent the links of the robot arms. The red voxels represent the distance field generated for self-collision checking.

compute a solution. All resulting trajectories were logged and metrics were computed on them. The averaged metrics are presented in the next section. All experiments were carried out on a quad-core Intel Core 2 Quad Q9550 CPU (2.83 GHz) with 5.8 GB of RAM running the Ubuntu Lucid variant of Linux and the Fuerte variant of ROS.

All of the planners were configured with the same collision checking library. Figure 8 shows the simplified representation of the PR2. The links of the robot that can be moved during the execution of a planned path are represented using a number of overlapping spheres. The remaining links as well as the environment are represented using a combination of a voxel grid and geometric primitives. The environment representation is processed to generate a distance field which is capable of providing the distance to the nearest obstacle from various points in the environment. The internal collision model of the robot can then be easily checked against this distance field to decide whether the robot is in collision with the environment. A separate check is performed to account for internal collisions. The distance field can also be used to represent an *obstacle cost* for robot configurations, measuring how far the robot is from an obstacle. A small amount of time is spent on generating the distance field while pre-processing the environment for collision checks. In our experiments, depending on the complexity of the environment, this can take anywhere between 100ms-600ms.

The configuration of our planner is as follows. MP_{lowD} contains 8 motion primitives. Each one rotates one of four joints (shoulder pan, pitch, roll and elbox flex) by $8°$ in either direction. MP_{fullD} contains 14 motion primitives, in which each motion primitive rotates one joint $4°$ in either direction. The grid resolution we used to compute the heuristic was 2 cm.

B. Performance Benchmarks

The metrics measured on each plan include the following:

1) planning time
2) planned length - the length of the path generated by the planner.
3) simplified length - the length of the path simplified during the post-processing step (e.g. shortcutting).
4) success rate

Table I shows the performance benchmarks for all the environments in Figure 9. The sampling-based planners are very fast but our approach is certainly competitive in most environments. In general, we found that the solutions generated by ARA* are noticeably shorter in path length. Note that in these experiments both ARA* and RRT* are only run until the first solution. Table II shows the results of ARA* and RRT* when given a time budget of 60 seconds for planning. Both planners were configured to minimize total path length in jointspace.

As stated earlier, the goals were defined as 6D poses for the end-effector. While ARA* is capable of planning to an end-effector pose, as configured, the sampling-based approaches are not. In our experiments, we ran ARA* to compute a valid joint configuration at the goal state and then use it as input to the sampling-based planners. Thus, the standard practice of first searching for a valid configuration at the goal pose using an inverse kinematics solver was not needed. This procedure can be time consuming especially if the goal pose is in a cluttered region of the workspace.

C. Consistency Benchmarks

For many planning 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 it. Planning with heuristic searches is typically very consistent, meaning that similar inputs generate similar outputs. To compare the consistency of the motions generated by the planners, we performed an experiment in which all three of the planners are called to plan from a single configuration of the robot to multiple goal poses that are within a close vicinity of each other. In Figure 10, the right arm of the PR2 is extended over the tabletop, and below the table are 27 6D goal poses, all of which are contained within a 10 cm cube. They are each given randomly generated orientation constraints that fall within the workspace of the arm. For these experiments, the planners were granted a translational tolerance of 0.5 cm and a rotational tolerance of 0.05 radians in each of roll, pitch and yaw. The planners planned to each goal only once. It can be argued that for the same start and goal conditions, sampling-based planners will return the same path every time if the randomization seed is fixed. However, by asking the planner to plan to *similar* (but not the same) goals, we can get a better idea of how consistently the planners will behave across multiple planning attempts.

To measure how spread apart the trajectories are, or how much distance there is between them, we compare the paths taken by the wrist and by the elbow along the planned path for the arm. We used three planners to plan 27 paths for the right arm of the PR2. We then compute the 3 dimensional *link paths* taken by the elbow and the wrist

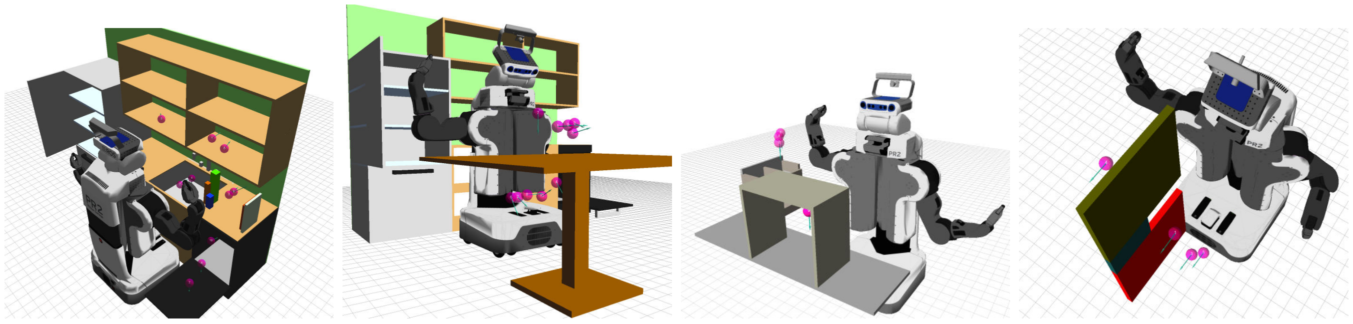

(a) Kitchen (b) Tabletop (c) Industrial (d) Narrow Passageway

Fig. 9: The pink spheres with cyan arrows indicate the desired 6D goal poses for the right end-effector.

Environment →	**Kitchen**			**Tabletop**			**Industrial**			**Narrow Passageway**		
Planner →	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*	ARA*	RRTC	RRT*
planning time *(mean, sec)*	0.31	0.01	0.87	0.98	0.01	0.03	0.14	0.01	6.06	0.74	0.66	3.90
planned length (joint space) *(mean, rad)*	9.52	13.13	12.90	10.97	10.20	10.19	5.76	12.12	12.33	17.33	25.66	23.24
simplified length (joint space) *(mean, rad)*	6.93	9.81	9.30	7.37	8.14	7.71	4.09	8.81	6.88	9.54	13.56	12.60
success rate	100%	100%	87%	100%	100%	100%	100%	100%	80%	100%	100%	100%

TABLE I: Performance comparison of three planners for single-arm manipulation in the scenarios shown in Figure 9.

Environment →	**Kitchen**		**Tabletop**		**Industrial**		**Narrow Passageway**	
Planner (60sec) →	ARA*	RRT*	ARA*	RRT*	ARA*	RRT*	ARA*	RRT*
planned length (joint space) *(mean, rad)*	7.74	9.21	6.51	10.71	4.43	14.26	12.58	23.18
success rate	100%	80%	100%	100%	100%	94%	100%	100%

TABLE II: Results of ARA* and RRT* with a fixed planning time budget of 60 seconds.

along those trajectories. The link paths for two planners (our approach and RRT-Connect) are shown in Figure 11. The red lines are the link paths for the wrist and in yellow are the link paths for the elbow corresponding to plans using our approach. The green and blue lines are the wrist and elbow link paths, respectively, for plans using RRT-Connect. The average lengths of these paths can be found in Table III.

We compute the distance between the link paths by discretizing each link path in the set into n waypoints. Then, for the ith waypoint in each path of the set, we compute its mean and variance. Finally, we sum up the variance of all n waypoints. To compute the variance listed in Table III, we used $n = 100$. The waypoints used for each path can be seen in Figure 12. Figure 13 is included to uncover the paths produced with our approach that are hidden below. The variance of the elbow link paths generated by our approach is approximately ten percent of that of RRT-Connect and RRT*. The variance of our wrist path is between a fourth and a fifth of RRT* and RRT-Connect.

VIII. DUAL-ARM EXPERIMENTAL RESULTS

A. Experimental Setup

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 for dual-arm manipulation. In cluttered environments, the workspace for dual-arm

same start → different goals	ARA*	RRTC	RRT*
length (joint space) *(mean, rad)*	8.828	23.565	22.259
length (joint space) *(std. dev., rad)*	2.758	14.705	17.438
length (wrist) *(mean, meters)*	1.921	2.831	2.831
length (wrist) *(std. dev., meters)*	0.255	1.929	2.365
length (elbow) *(mean, meters)*	1.133	1.769	1.703
length (elbow) *(std. dev., meters)*	0.155	1.050	1.469
variance (wrist) *(total, meters^2)*	11.721	124.085	104.662
variance (elbow) *(total, meters^2)*	10.128	55.716	44.023

TABLE III: Results from 27 trials with the same initial configuration and with different goals (see Figure 10). The length refers to the planned length.

manipulation is even smaller. To generate benchmark tests in this domain, 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 15 shows the different simulation environments. The obstacles are in purple and the collision model of the manipulated objects can be seen in cyan. Stills of the robot during the actual experiments on

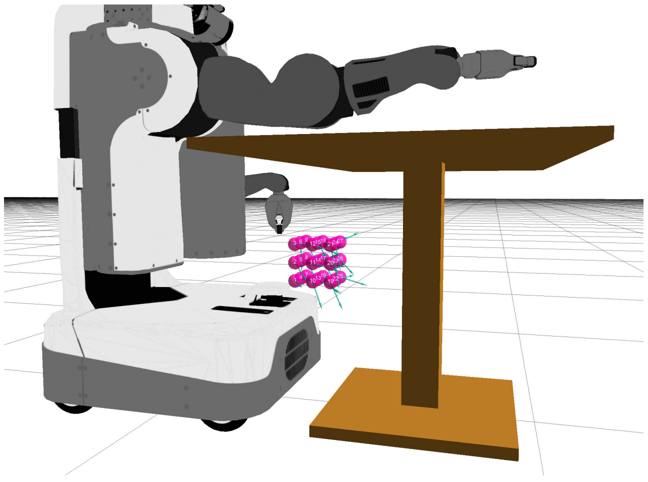


Fig. 10: To demonstrate the consistency of the set of planners we are comparing, the planners are asked to plan for the right arm from the position above the table to the 27 different goal poses beneath it. The 27 equidistant poses are within a 10 cm cube and have randomly generated orientations.

the PR2 can be seen in Figure 16.

For all of our dual-arm experiments, we used a set of 32 static motion primitives. The set includes 26 motions in which each one translates the object one cell in the direction of one of the edges in a 26-connected grid. Our set also includes four motions that rotate the redundant joints. Each one of these motions rotates one of the two redundancies in either direction. Lastly, our set includes two motions that just yaw the object in the world frame. While this set of very basic motion primitives does provide a dense coverage of the workspace, in the future we plan on researching methods of constructing motion primitives that are smooth, dense and efficient. A summary of the motions can be seen in Figure 14.

B. Performance Benchmarks

The results of the simulated experiments are shown in Table IV. 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.

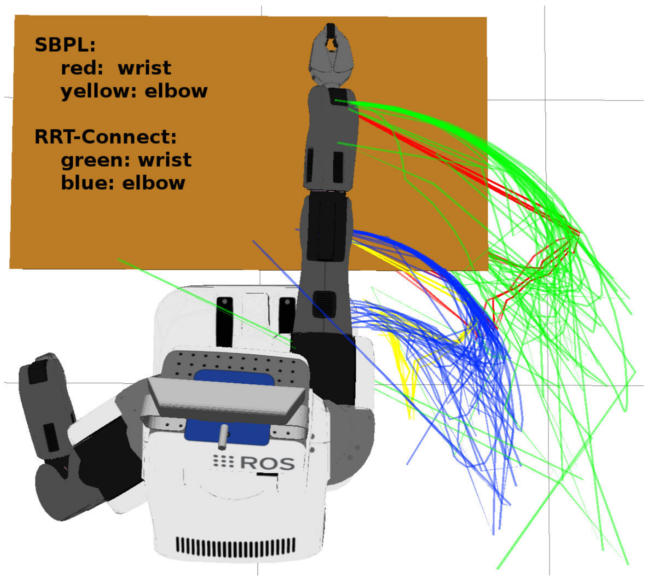


Fig. 11: The paths taken by the elbow and the wrist during the planned trajectories are shown above. The red(wrist) and yellow(elbow) paths were planned using our approach. The blue(elbow) and green(wrist) paths were planned by RRT-Connect.

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 IV: Results from 12 simulated trials.

IX. DISCUSSION

Algorithmic Parameters. Our algorithm has a couple of parameters that can affect its performance, such as the grid resolution in which we compute h_{BFS} and d_{ik}. We have found a grid resolution of 2 cm provides an informative heuristic and does not restrict the planner in any of our experiments. Experimentally we determined that a good value for d_{ik}, as defined in Section III, is 6 cm.We experimented with values of d_{ik} ranging from 40 cm down to 2 cm.

Motion Primitives. In Sections VII and VIII, we briefly describe the sets of motion primitives that we use in our experiments. In summary, our approach has generally been to use sets of motion primitives, each of which move in a single dimension, either jointspace or workspace. While it proves to be adequate for many tasks, it is future work to explore methods of generating motion primitives that change more than one joint at a time. An example of such a primitive is a motion that simultaneously rolls the forearm clockwise by $2°$ while flexing the elbow by $8°$. In particular, it would be useful to look for inspiration from principled methods for generating motion primitives for navigation [20] [21].

Heuristic. Evident by our experimental results, h_{BFS} is

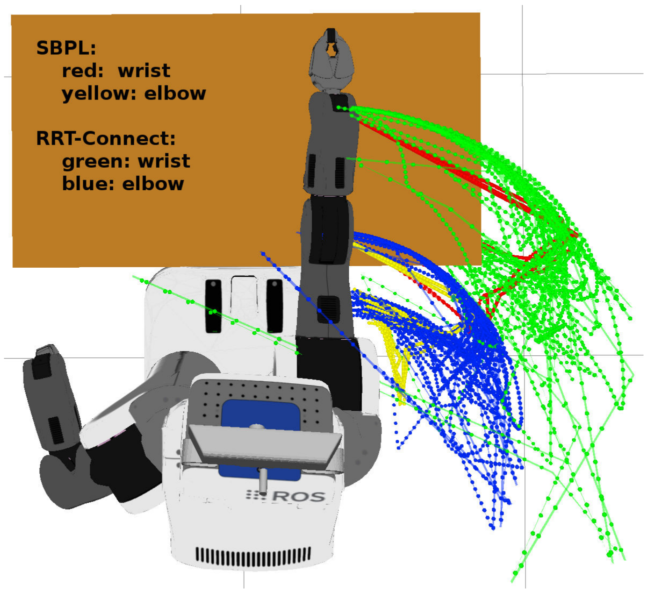


Fig. 12: The planned paths lined with the equidistant waypoints used to compute the distance between each set of paths. The variance is computed for each waypoint index and then summed. The total variance for each planner and link path combination is shown in Table III.

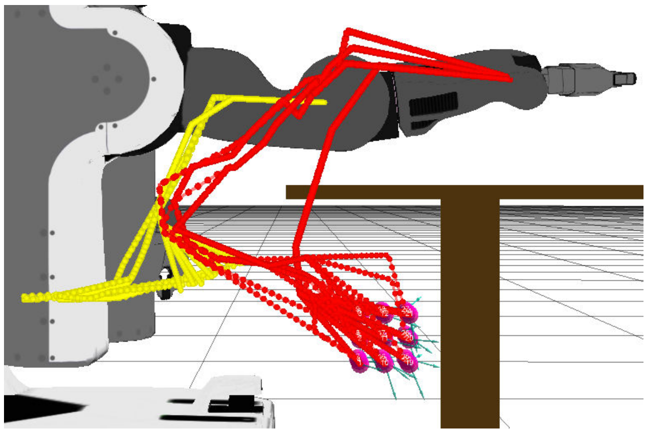


Fig. 13: A closeup of the paths computed with our approach (red: wrist, yellow: elbow). Notice that many of the paths overlap each other and diverge just beneath the table towards their respective goals.

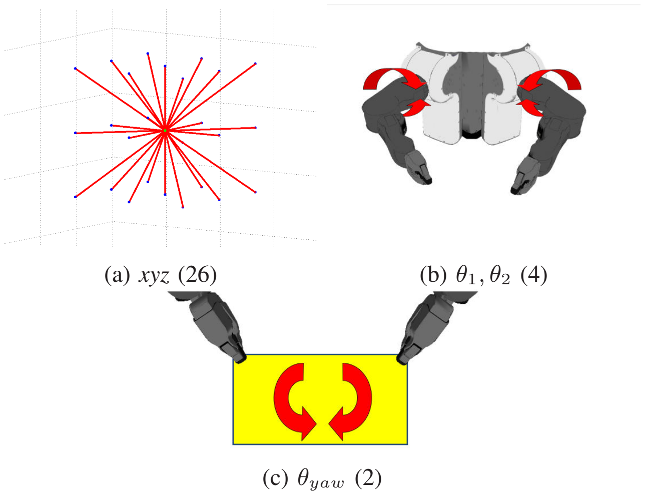


Fig. 14: The set of 32 static motion primitives we used during our experiments.

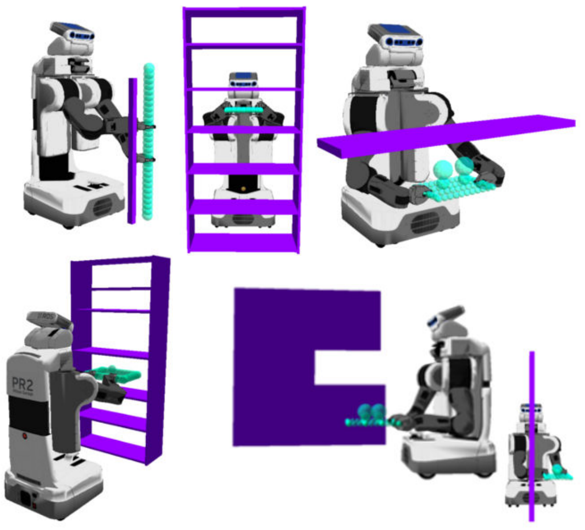


Fig. 15: Clockwise from top left: stick around a pole, wood board in bookshelf, tray with wine glasses under a table, tray with wine glasses near wall and tray with a scotch glass in bookshelf.

very informative with avoiding local minima due to obstacles. However, as mentioned in Section III, situations exist in which h_{BFS} can still be trapped by local minima. Given that the kinematics of the arm are ignored and the end-effector (or object) is approximated as an untethered sphere (or cylinder), the shortest path computed by h_{BFS} may prove infeasible for the robot to perform for one of two reasons. Either because the shortest path exits the manipulation workspace of the robot at some point or the shortest path found is in a class in which no feasible path for the robot exists. An example of the second problem can be seen in Figure 17. A simple solution to the first problem is to mark all cells in the grid as invalid that are beyond the workspace of the arm before computing h_{BFS}. In [6] we proposed a solution to the second problem in which we combine h_{BFS} with a second heuristic called h_{elbow}. For a given state, h_{elbow} represents the shortest path from the current pose of the elbow link to any of the feasible poses where the elbow can reside while keeping the end-effector at the goal pose. In the problematic situations, the sum of the two heuristics was very successful in avoiding the large local minima. It is future work to determine how to combine the heuristics more effectively when the problem does not exist.

Discretization. Search-based planning relies on the discretization of the state-space and the action space. There are

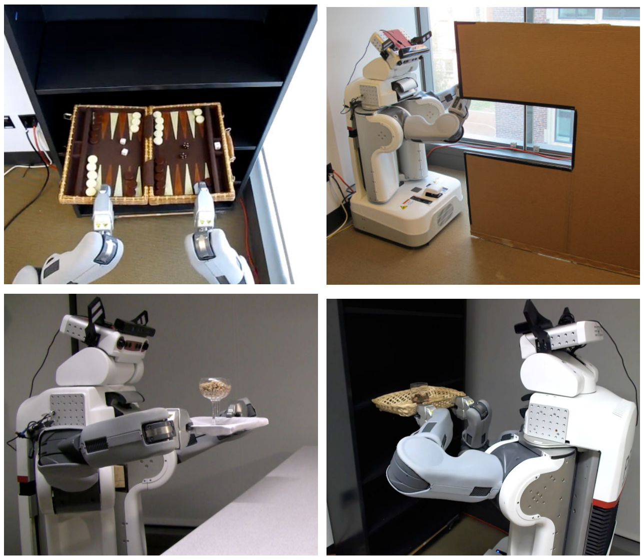


Fig. 16: Shown here are four of the experiments that were run on the PR2.

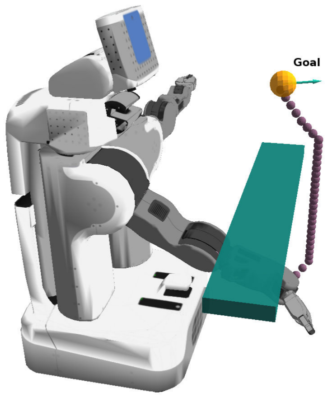


Fig. 17: h_{BFS} suggests a path to the goal that is infeasible for the robot to follow.

pros and cons to it. On one hand, the discretization can lead to a motion that looks somewhat discretized. On the other hand, a simple deterministic shortcutting routine can deal effectively with this artifact. In fact, it has an advantage over common methods of path simplification for sampling-based approaches in that the solutions generated by our search-based approach allow a shortcutter to do most of the work in a single pass through the points, and the second iteration provides little benefit. In contrast, shortcutters for sampling-based planners typically require many iterations [10]. In our experiments, the complete post-processing time per plan was typically around 20ms. Furthermore, the representation in the workspace plus redundant degree of freedom (such as the one explained in Section V) typically results in a motion that does not require any shortcutting at all.

Theoretical Guarantees. Our approach is based on using a heuristic search with strong theoretical guarantees to compute a high quality solution. It is important to note that those guarantees are with respect to the graph constructed, not to the continuous motion planning problem. The variable dimensionality, non-uniform resolution, and the set of motion primitives that we use, all play a part in determining whether a solution can even be found in our graph and its quality. In the future, it would be interesting to research how to simultaneously provide guarantees on completeness and optimality w.r.t. to our graph construction as well as regarding the continuous planning problem.

X. Conclusion

We have presented a heuristic search-based approach to motion planning for manipulation that leverages the construction of a manipulation lattice graph, informative heuristics and an anytime graph search to deal effectively with various manipulation planning problems. The manipulation lattice graph relies on the use of adaptive motion primitives and non-uniform dimensionality and non-uniform resolution of the lattice to plan efficiently and precisely. In addition to its explicit cost minimization. Our approach benefits from the use of an anytime graph search to generate consistent and good quality solutions quickly, as well as to provide theoretical guarantees on the completeness and bounds on the suboptimality of the solution cost, both with respect to the constructed graph. The efficiency of the search is aided heavily by an informative heuristic that dissipates the deep local minima caused by environmental complexities. We presented comparisons of performance, quality and consistency between our approach and several sampling-based planning approaches on practical planning problems on a PR2 robot. The results show that our approach can plan in many complex situations with sub-second planning times. Finally, our experimental analysis shows that due to its deterministic cost-minimization, the approach generates motions that are of good quality and are consistent, i.e. the resulting plans tend to be similar for similar tasks. 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 controlling or interacting with the robot.

In future work, we are examining extensions of this approach to higher dimensional problems like full-body planning. In particular, we believe that this approach, when combined with a *caching* or *learning* algorithm that attempts to reuse prior experience, can be successful even in higher dimensions.

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

[3] Ben Cohen, Ioan A. Şucan, and Sachin Chitta. A generic infrastructure for benchmarking motion planners. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 589–595, Vilamoura, Portugal, October 2012.

[4] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for dual-arm manipulation with upright orientation constraints. In *IEEE Int. Conference on Robotics and Automation*, St. Paul, Minnesota, 2012. IEEE.

[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] Ioan Şucan, Mark Moll, and Lydia Kavraki. The Open Motion Planning Library (OMPL). http://ompl.kavrakilab.org, 2010.

[8] 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.

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

[10] R. Geraerts and M.H. Overmars. Clearance based path optimization for motion planning. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2386 – 2392 Vol.3, april-1 may 2004.

[11] 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.

[12] 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.

[13] 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.

[14] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *International Conference on Robotics and Automation*, Shanghai, China, May 2011.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] M. Pivtoraiko and A. Kelly. Generating state lattice motion primitives for differentially constrained motion planning.

[21] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2172 – 2179, sept. 2011.

[22] 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.

[23] 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.