Planning with Adaptive Dimensionality for Mobile Manipulation

Kalin Gochev
University of Pennsylvania

Alla Safonova
University of Pennsylvania

Maxim Likhachev
Carnegie Mellon University

***Abstract*— Mobile manipulation planning is a hard problem composed of multiple challenging sub-problems, some of which require searching through large high-dimensional state-spaces. The focus of this work is on computing a trajectory to safely maneuver an object through an environment, given the start and goal configurations. In this work we present a heuristic search-based deterministic mobile manipulation planner, based on our recently-developed algorithm for planning with adaptive dimensionality. Our planner demonstrates reasonable performance, while also providing strong guarantees on completeness and suboptimality bounds with respect to the graph representing the problem.**

Keywords: Mobile Manipulation Planning, Planning Algorithms, Heuristic Search

I. Introduction

In recent years robotics research has moved from the controlled predictable industrial environments towards the cluttered, uncontrolled and unpredictable domestic environments, where robots need to be able to safely perform a variety of tasks. These tasks often involve manipulating various objects. As a result, manipulation planning for mobile robots is a popular research field, which presents many challenges due to its inherently high-dimensional nature. Planners for mobile manipulation are required to produce synchronized trajectories for both the robot's movable base and its manipulator, both of which have multiple degrees of freedom.

Mobile manipulation planning can be broken down into several sub-problems such as planning how to grasp an object or put it down, and maneuvering it safely through the environment to a desired location. The focus of this work is on the latter. Namely, the motion planning for the robot's manipulator and movable base from a given start configuration to a given goal configuration.

Search-based planning algorithms are often used in many areas of robotics, such as navigation planning [1], [2]. There are several reasons for the popularity of search-based planners. First, they typically provide strong guarantees on completeness and bounds on suboptimality. Second, a number of anytime search algorithms have been developed, that find the best solution they can within a given time [3], [4], [5]. Third, a number of search algorithms can re-use previous search efforts to find new solutions faster [6], [7]. Finally, treating the problem as a cost-minimization problem allows one to formulate and incorporate complex cost functions and constraints into the planning process. Although search-based planners have been shown to work well for static manipulation tasks [8], they typically have not been used for mobile manipulation planning for high-DOF robotic manipulators, due to the high dimensionality of the planning problem. In this paper, we present a heuristic search-based planner utilizing our previously-developed algorithm for planning with adaptive dimensionality, which employs dimensionality-reduction techniques and an informative heuristic to speed up the search process. Our planner demonstrates reasonable performance, while also providing provable guarantees on completeness and solution suboptimality bounds with respect to the graph that encodes the problem.

This research was sponsored by ONR grant N00014-09-1-1052, DARPA CSSG program D11AP00275 and the Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0016.

II. Related Work

The most common approaches to solving mobile manipulation planning problems have been based on the probabilistic sampling-based planning algorithms such as probabilistic roadmaps [9], RRT [10] and its variants [11], [12], [13]. These planners perform extremely well in most instances and are able to very quickly identify feasible solutions to high-dimensional planning problems. However, their performance can suffer significantly in very cluttered environments with narrow solution spaces. Moreover, sampling-based planners are focused on finding any feasible trajectory, rather than minimizing the cost of the solution. As such, they may often produce solutions of unpredictable length involving jerky motions that may be hard for the manipulator to follow and the solutions can be inconsistent from one planning episode to another. To remedy this problem, various smoothing techniques have been used. Although often helpful, smoothing may fail in highly cluttered environments. In addition, sampling-based planning algorithms only provide probabilistic guarantees on completeness and solution suboptimality bounds. A notable exception is the RRT* algorithm [14], which asymptotically converges to an optimal solution.

Researchers have also developed behavior-based approaches for mobile manipulation [15], [16]. Those techniques utilize state-machines and pre-scripted behaviors to complete a given manipulation task and usually do little or no high-dimensional planning. Such approaches are usually targeted at completing very specific tasks, such as opening of doors, and don't provide a robust framework for solving general mobile manipulation problems. These approaches do not provide any guarantees on completeness or solution suboptimality bounds.

Some approaches decrease the dimensionality of the problem by planning in the joint state-space of the object being manipulated and the robot's base, and then running an

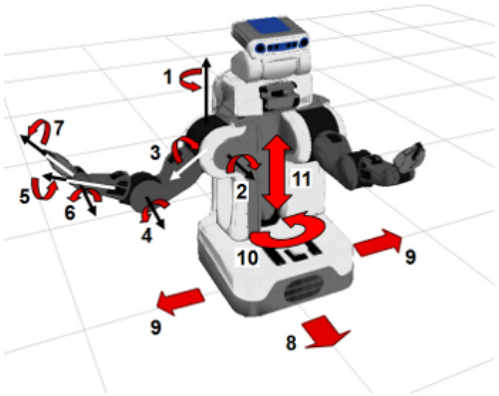


Fig. 1: The 11-DOF of the PR2 robot used by our planner. (1: shoulder pan; 2: shoulder lift; 3: shoulder roll; 4: elbow flex; 5: forearm roll; 6: wrist flex; 7: wrist roll; 8,9,10: base XY position and heading; 11: torso height)

inverse-kinematics solver to produce trajectories for the base and the manipulator [17]. However, these approaches do not perform well in cluttered environments, in which the computed arm motions may not be feasible for execution by the robot.

Our approach also reduces the dimensionality of the problem in order to decrease planning times, but it does so by constructing and searching a significantly smaller state-space, consisting of both high-dimensional and low-dimensional states. Our planner differs from other mobile manipulation planners in that it uses a deterministic search algorithm and provides provable deterministic guarantees on completeness and plan suboptimality bounds with respect to the graph representing the problem.

III. Problem Definition

We are assuming that the planning problem is represented by a discretized finite state-space S of dimensionality d and a set of transitions $T = \{(X_i, X_j) | X_i, X_j \in S\}$. Each pair $(X_i, X_j) \in T$ corresponds to a feasible transition between the corresponding state vector values and is associated with a positive cost $c(X_i, X_j)$. We will use the notation $\pi(X_i, X_j)$ to denote a path from state X_i to state X_j, and its cost will be denoted by $c(\pi(X_i, X_j))$. We will use $\pi^*(X_i, X_j)$ to denote a least-cost path. Thus, we have an edge-weighted graph G with a vertex set S and edge set T. The goal of the planner is to find a least-cost path in G from a given start state X_S to a goal state X_G. Alternatively, given a desired suboptimality bound $\epsilon \geq 1$, the goal of the planner is to find a path $\pi(X_S, X_G)$, such that $c(\pi(X_S, X_G)) \leq \epsilon \cdot c(\pi^*(X_S, X_G))$.

We used Willow Garage's PR2 robot as the testing platform for our planner. The robot has two humanoid arms, movable base, and adjustable torso height. The full arm configuration on the PR2 is given by its seven joint angles (shoulder pan, shoulder lift, shoulder roll, elbow flex, forearm roll, wrist flex, wrist roll). We included only the right arm in the planning process. The base and torso provided additional 4 degrees of freedom: base XY position and heading, and torso height. Thus, our domain had a total of 11 degrees of freedom (Fig. 1).

IV. Planning with Adaptive Dimensionality

In this section we will provide a brief overview of our algorithm for planning with adaptive dimensionality. For a more detailed explanation of the algorithm, we refer the reader to our previous work [18].

While planning in a high-dimensional state-space is often necessary, large portions of the computed paths have a lower-dimensional structure. Planning for robotic arm manipulation, for example, can often be reduced to 3D planning for the end-effector and then running an inverse kinematics solver to find the full-dimensional path that corresponds to the computed end-effector trajectory. At the same time, there are relatively infrequent situations when the planner does need to consider the full configuration of the arm in trying to figure out the feasibility of the end-effector path. Based on this observation, our planning algorithm iteratively constructs a state-space S^{ad} and corresponding transition set T^{ad} consisting mainly of low-dimensional states and transitions, and only introducing regions of high-dimensional states and transitions into the state-space where it is necessary in order to ensure the feasibility of the resulting path.

The planning algorithm considers two state-spaces—a high-dimensional S^{hd} with dimensionality h, and a low-dimensional S^{ld} with dimensionality l, which is a projection of S^{hd} onto a lower dimensional manifold ($h > l, |S^{hd}| > |S^{ld}|$). We define a many-to-one mapping

$$\lambda : S^{hd} \rightarrow S^{ld}$$

from the high-dimensional state-space S^{hd} to the low-dimensional state-space S^{ld}. We also define the mapping $\lambda^{-1} : S^{ld} \rightarrow (S^{hd})^*$ from the low-dimensional state-space S^{ld} to subsets of the high-dimensional state-space S^{hd} as follows:

$$\lambda^{-1}(X^{ld}) = \{X \in S^{hd} | \lambda(X) = X^{ld}\}$$

Each of the two state-spaces may have its own set of transitions. However, we require that the costs of the transitions be such that for every pair of states X_i and X_j in S^{hd},

$$c\left(\pi^*\left(X_i, X_j\right)\right) \geq c\left(\pi^*\left(\lambda(X_i), \lambda(X_j)\right)\right)$$

We require that the cost of a least-cost path between any two states in the high-dimensional state-space to be at least the cost of a least-cost path between their images in the low-dimensional state-space. One can choose to use any two cost functions for T^{hd} and T^{ld} as long as they satisfy the above constraint.

Let G^{hd} and G^{ld} represent the corresponding graphs defined by S^{hd} and S^{ld} and their respective transition sets T^{hd} and T^{ld}.

In the case of 11-DOF mobile manipulation planning, S^{hd} consisted of 11-dimensional state vectors describing the full configuration of the robot and S^{ld} consisted of 3-dimensional states representing the end-effector XYZ position.

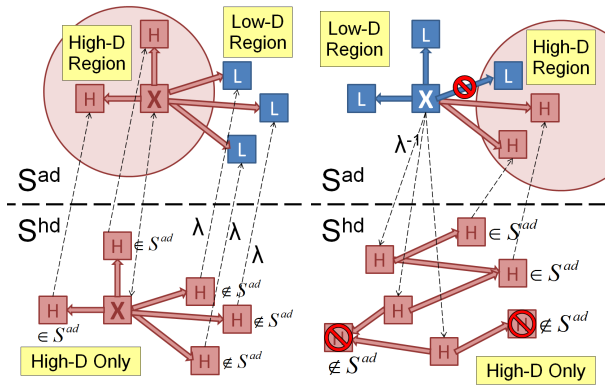


Fig. 2: The figure illustrates how to generate the successors of a high-dimensional state (left) and a low-dimensional state (right) in S^{ad}. States labeled with X are the states for which transitions are being generated. High- and low-dimensional states are labeled with H and L, respectively. States connected by dashed arrows represent equivalent states or their projections in the other dimensionality. Block arrows represent transitions between states. Transitions between two low-dim. states are low-dimensional. All other transitions are high-dimensional.

A. Algorithm

Structure of S^{ad}: Recall that the goal of our algorithm is to use the faster low-dimensional planning, except for areas of the environment where high-dimensional planning is necessary to ensure the feasibility of the resulting path. We want our adaptively-dimensional state-space to capture this property—namely, we want to have largely low-dimensional states in S^{ad}, except for the areas where high-dimensional planning needs to be done, represented by high-dimensional states and transitions in S^{ad}. In the low-dimensional areas of S^{ad} we can use simpler low-dimensional transitions. However, recall that the transitions we have in T^{hd} and T^{ld} connect two states of the same dimensionality, which do not allow us to transition between states of different dimensionalities. Therefore, we have to construct a transition set T^{ad} that allows for such transitions.

Construction of S^{ad}: Our algorithm iteratively constructs S^{ad}, beginning with the low-dimensional-state space S^{ld} and introducing a set of high-dimensional regions R in it. We will first explain how the high-dimensional regions are being introduced into S^{ad} and connected with the low-dimensional regions. The algorithm that decides when and where to introduce these regions will be explained later.

Once a high-dimensional region $r \in R$ is introduced, the following changes are made to S^{ad}. All low-dimensional states X_i^{ld} that fall inside $r \in R$ are replaced with their high-dimensional projection states in $\lambda^{-1}(X_i^{ld})$. Notice that if a high-dimensional state X^{hd} is in S^{ad}, then its low-dimensional projection $\lambda(X^{hd})$ is not in S^{ad}, and also if $X^{hd} \notin S^{ad}$, then $\lambda(X^{hd}) \in S^{ad}$.

Next we define the transition set T^{ad} for the adaptively-dimensional state-space as follows (Fig. 2). For any state $X_i \in S^{ad}$:

- If X_i is high-dimensional then for all high-dimensional transitions $(X_i, X_j^{hd}) \in T^{hd}$, if $X_j^{hd} \in S^{ad}$ then $(X_i, X_j^{hd}) \in T^{ad}$. If $X_j^{hd} \notin S^{ad}$, then $(X_i, \lambda(X_j^{hd})) \in T^{ad}$. That is, for high-dimensional states we allow only high-dimensional transitions to other high-dimensional states if they fall inside S^{ad}, or their respective low-dimensional projections (Fig. 2 left).
- If X_i is low-dimensional then for all low-dimensional transitions $(X_i, X_j^{ld}) \in T^{ld}$, if $X_j^{ld} \in S^{ad}$ then $(X_i, X_j^{ld}) \in T^{ad}$, and for all high-dimensional transitions $(X, X_j^{hd}) \in T^{hd}$, where $X \in \lambda^{-1}(X_i)$, if $X_j^{hd} \in S^{ad}$ then $(X_i, X_j^{hd}) \in T^{ad}$. That is, for low-dimensional states we allow low-dimensional transitions if they lead to another low-dimensional state in S^{ad}, and high-dimensional transitions from all of their high-dimensional projections if they lead to a high-dimensional state in S^{ad} (Fig. 2 right).

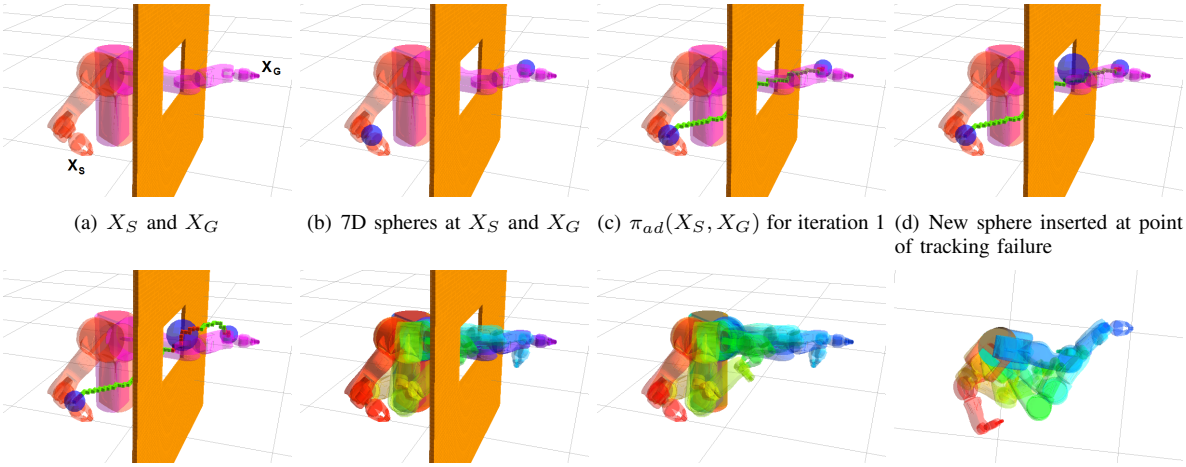
Notice, that the above definition of T^{ad} allows for transitions between states of different dimensionalities.

The adaptively-dimensional state-space S^{ad} and the transition set T^{ad} give us a graph G^{ad} of adaptive dimensionality. Adding new high-dimensional regions or increasing the sizes of existing regions require the reconstruction of S^{ad} and T^{ad}, and thus, will produce a new instance of G^{ad}.

We also define a tunnel τ of radius w around an adaptively-dimensional path π_{ad} as follows: τ is a subgraph of G^{hd}, and thus consists entirely of high-dimensional states and transitions. A high dimensional state $X^{hd} \in \tau$ if there exists a state $X_i \in \pi_{ad}$ such that the distance from $\lambda(X^{hd})$ to X_i (or $\lambda(X_i)$ if X_i is high-dimensional) is no larger than w, for some pre-defined distance metric in S^{ld}. We include all transitions (X_j, X_k) from T^{hd} such that both X_j and X_k are in τ.

We continue this section with an intuitive description of our proposed algorithm. Algorithm 1 gives the pseudo code for our algorithm. Each iteration of the algorithm consists of two phases—an adaptive planning phase and a path tracking phase. In the adaptive planning phase, the current instance of G^{ad} is searched for a least-cost path from start to goal. The tracking phase, then attempts to construct a high-dimensional executable path to match (or track) the adaptive path computed in the adaptive planning phase.

Initially, G^{ad} is the same as G^{ld}, with two high-dimensional regions added around the start and goal states (Algorithm 1, lines 1-3), which are necessary since the given start and goal states are high-dimensional. At each iteration, a new instance of G^{ad} is constructed based on the set of high-dimensional regions, and is searched for a least-cost path π_{ad}^* from X_S to X_G. Notice that π_{ad}^* consists of both low-dimensional and high-dimensional states, so it is not an executable path. If no path is found in the adaptive planning phase, then no feasible path exists from start to goal and the algorithm terminates. If an adaptive path π_{ad}^* is found, then the path tracking phase constructs a tunnel τ of radius w around the adaptive path π_{ad}^*. Then τ is searched for a least-cost path π_τ^* from start to goal. Note that since τ



(a) X_S and X_G (b) 7D spheres at X_S and X_G (c) $\pi_{ad}(X_S, X_G)$ for iteration 1 (d) New sphere inserted at point of tracking failure

(e) $\pi_{ad}(X_S, X_G)$ for iteration 2 (f) Final 7D arm trajectory after successful tracking (g) Final trajectory (obstacles not shown) (h) Final trajectory (top view)

Fig. 3: Example environment for 7D robotic arm motion planning (robot's base and torso remain stationary). A trajectory is computed of how the arm can be maneuvered from the start configuration to reach through the opening to the goal arm configuration in two iterations of our algorithm. 3(c) and 3(e) show the adaptively-dimensional paths computed at each iteration.

consists of only high-dimensional states and transitions, π^*_τ is a fully high-dimensional path, and thus, it is executable. If no path is found in τ, then a new high-dimensional region is introduced in G^{ad} or the sizes of the existing regions are increased, and the algorithm proceeds to the next iteration. If a path is found in τ, but its cost does not satisfy the cost constraint (e.g. $c(\pi^*_\tau) > \epsilon_{track} \cdot c(\pi^*_{ad})$), then a new high-dimensional region is introduced or the sizes of existing high-dimensional regions are increased, and another iteration is started. If $c(\pi^*_\tau) \leq \epsilon_{track} \cdot c(\pi^*_{ad})$, then the algorithm returns π^*_τ as a feasible path from start to goal and terminates. The returned path is guaranteed to have cost that is no more than ϵ_{track} times the cost of an optimal path in G^{hd}.

Identifying the places where high-dimensional regions need to be introduced is a non-trivial problem in itself. In our experiments, the search within the tunnel during the path tracking phase keeps a record of how far along the tunnel states have been expanded. Thus, if the search in τ fails, we are able to reconstruct a path to the point where the search had failed, and we introduce a new high-dimensional region there. If the algorithm successfully finds a path through the tunnel, but the path is too costly compared to the path found in G^{ad}, then we take the following approach in identifying locations where new high-dimensional regions need to be introduced. We approximate the location, where the largest cost discrepancy between π^*_{ad} and π^*_τ is observed, by going along both paths simultaneously and comparing the cumulative costs observed so far. We identify all points of large cost discrepancy and introduce new high-dimensional regions there. This approach tends to remedy the cost discrepancy, and generally works well in identifying the regions that require high-dimensional planning.

B. *Theoretical Properties*

We have shown the following theoretical properties of our algorithm for planning with adaptive dimensionality in [18].

Theorem 4.1: If we have a finite state-space, Algorithm 1 terminates and at the time of its termination, the cost of the returned path $\pi(X_S, X_G)$ is no more than ϵ_{track} times the cost of an optimal path from state X_S to state X_G in G^{hd}.

Theorem 4.2: If ϵ_{search}-suboptimal graph search algorithm is used in line 5 of Algorithm 1, the cost of the path returned by our algorithm is no larger than $\epsilon_{search} \cdot \epsilon_{track} \cdot \pi^*_{hd}(X_S, X_G)$.

Figure 3 illustrates the adaptively dimensional planning process in the case of 7-DOF robotic arm manipulation planner, planning adaptively in 7D/3D. Low-dimensional 3D states represent the end-effector XYZ position, while high-dimensional 7D states represent the full arm configuration. High-dimensional regions are represented as spheres.

V. Application of Planning with Adaptive Dimensionality to Mobile Manipulation

In the case of the robotic arm motion planning, our goal was to use a 11D/3D adaptive planning, where 3D states represented the arm's end-effector XYZ position, and 11D states represented the full arm, torso and base configurations. As the full arm configuration on the PR2 robot is given by its seven joint angles, constructing a projection function λ mapping full joint angle configuration to end effector position presented several challenges—namely discretization of the joint angle space could not be easily matched to a discretization of the end-effector position space, and λ and λ^{-1} would have needed to involve expensive FK and IK computations. Instead, we decided to transform the standard 7-DOF robot arm configuration representation to one described in [19], which converts joint angles representations

Algorithm 1 Path Planning with Adaptive Dimensionality

$G^{ad} = G^{ld}$
AddFullDimRegion($G^{ad}, \lambda(X_S)$)
AddFullDimRegion($G^{ad}, \lambda(X_G)$)
loop
  search G^{ad} for least-cost path $\pi^*_{ad}(X_S, X_G)$
  if $\pi^*_{ad}(X_S, X_G)$ is not found **then**
    return no path from X_S to X_G exists
  construct a tunnel τ around $\pi^*_{ad}(X_S, X_G)$
  search τ for least-cost path $\pi^*_{\tau}(X_S, X_G)$
  if $\pi^*_{\tau}(X_S, X_G)$ is not found **then**
    let $\pi(X_S, X_{end})$ be the returned path
    if X_{end} is already within FullDimRegion in G^{ad} **then**
      GrowFullDimRegion($G^{ad}, \lambda(X_{end})$)
    else
      AddFullDimRegion($G^{ad}, \lambda(X_{end})$)
  else if $c(\pi^*_{\tau}(X_S, X_G)) > \epsilon_{track} \cdot c(\pi^*_{ad}(X_S, X_G))$ **then**
    identify a state X_r where a new FullDimRegion needs to be introduced
    if X_r is already within FullDimRegion in G^{ad} **then**
      GrowFullDimRegion(G^{ad}, X_r)
    else
      AddFullDimRegion(G^{ad}, X_r)
  else
    return $\pi^*_{\tau}(X_S, X_G)$

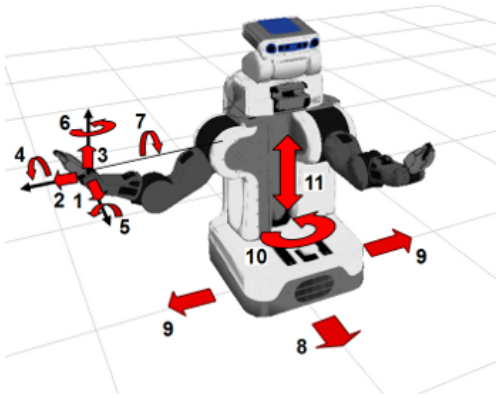


Fig. 4: The 11-DOF of our planner showing the alternative 7-DOF arm representation. (1,2,3: end-effector XYZ position; 4,5,6: end-effector RPY orienation; 7: arm swivel angle; 8,9,10: base XY position and heading; 11: torso height)

of a 7-DOF arm to 7-DOF representation consisting of the following values: (end-effector x position, end-effector y position, end-effector z position, end-effector roll, end-effector pitch, end-effector yaw, arm swivel angle) (Fig. 4). For more details on the representation, consult [19]. This alternative representation of the full arm configuration did not change the dimensionality of the high-dimensional state-space, but provided clean and easy λ and λ^{-1} mappings without any dicretization inconsistencies.

Using this alternative representation, our 11-dimensional states were represented by the following state vector:

$$(ee_{position}, ee_{orientation}, swivel, base, torso_{height}),$$

where ee_{pos}, ee_{ori} and $base$ consist of 3 values each—end-effector XYZ, end-effector RPY, and base XY and heading,

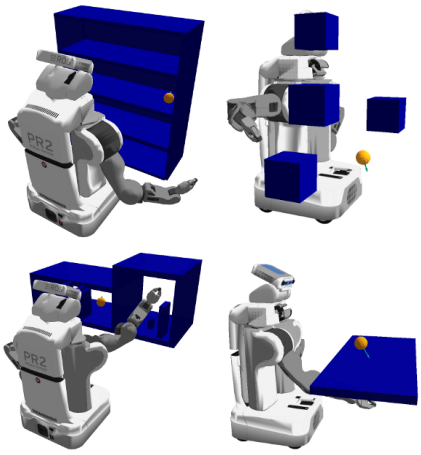


Fig. 5: Example environments used in our simulations

respectively. We used the following mapping functions:

$$\lambda(ee_{pos.}, ee_{ori.}, swivel, base, torso_{ht}) = (ee_{pos})$$

$$\lambda^{-1}(ee_{pos}) = \{(ee_{pos}, ee_{ori}, swivel, base, torso_{ht})|$$

for all feasible values of $ee_{ori}, swivel, base$ and $torso_{ht}\}$

The end-effector was allowed to move in a 3m×3m×2m 3D uniform grid with resolution of 2cm, centered around the robot. We used 6cm resolution for the base position and torso height. We uniformly discretized the values for the end-effector roll, pitch and yaw angles, the arm swivel angle, and the base heading angle into 16 on the interval $(-\pi, \pi]$. This discretization produced a 3D grid for the end-effector of size 150×150×100, or roughly 2.25×10^6 low-dimensional states. Our high-dimensional state-space consisted of about 1.8×10^{15} states.

We used very simple motion primitives for graph transitions for the motion planning—namely we allow ± 1 change in each of the eleven discretized state-vector values. This produces 22 possible transitions for 11D states and 6 possible transitions for 3D states. Due to the simplicity of the motion primitives, the resulting arm trajectory is not very smooth, but experimenting with a more complex set of motion primitives is one of our future work goals.

The cost of each low-dimensional motion primitive was representative of the distance traveled by the end-effector when executing that primitive. The costs of high-dimensional motion primitives included the distance traveled by the base and penalties for changes in any of the angular values of the state, as well as the distance traveled by the end-effector.

Obstacles in the environment are obtained through a collision map produced by the tilting laser scanner of the PR2. Very basic collision-checking is performed on low-dimensional states, treating them as point-robots and checking them against the obstace map. Full collision checking is performed on high-dimensional states, checking the full robot configuration (arm, torso, and base) against the obstacle map, while also enforcing joint-limits on the arm configuration. States that are found to be in collision during the

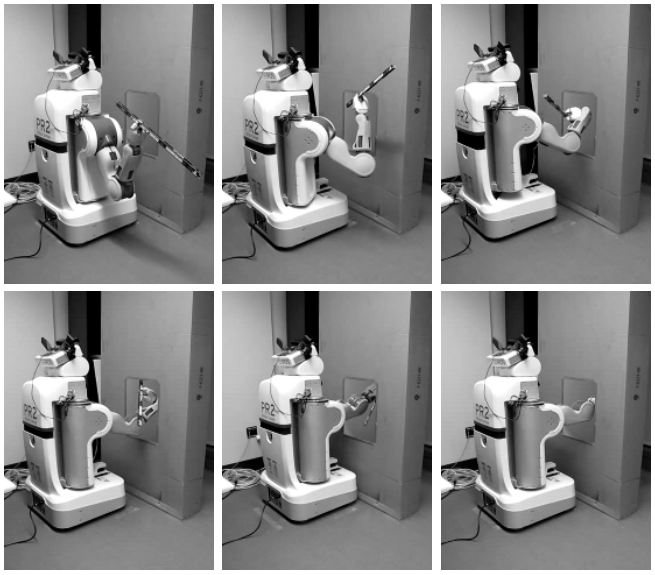


Fig. 6: PR2 manipulating 80cm stick trough a 40cm×50cm window.

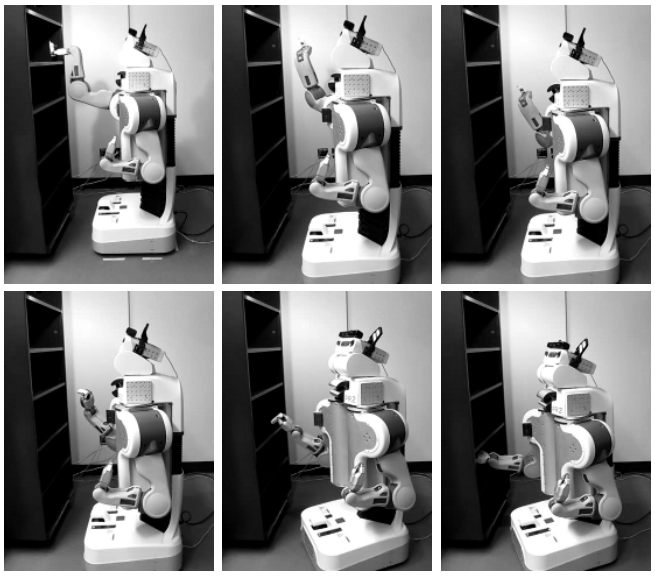


Fig. 7: PR2 reaching from a high shelf to a low shelf of a bookcase

search are discarded from G^{ad}. Recall that the path returned by our algorithm consists of only high-dimensional states, on which full collision-checking has been performed, and thus are collision-free.

The graph search algorithm we used for both the adaptive planning and the path tracking phases was Anytime Repairing A* (ARA*) [3], which provides provable completeness and suboptimality guarantees, and as such, satisfies Theorem 4.2.

VI. Results

A. Simulation

We ran the 11D/3D adaptive dimensionality planning algorithm, a full 11D planning algorithm (using weighted A* without re-expansions as desribed in [3] to search the full 11D state-space), and an 11D bi-directional RRT algorithm [11] on 30 environments in simulation and compared the results. Environments ranged in degree of difficulty—some required very simple motions to navigate from start to goal, while others were more cluttered and required a set of complex maneuvers to navigate around the obstacles. Some of the types of environments we used included various table tops, bookshelves, and random cuboid obstacles (Fig. 5). Both the adaptively-dimensional and the 11D planners utilized a 3D Dijkstra heuristic to guide the planners to the position constraint. We treated the end-effector as a point robot of radius equal to the radius of the smallest link of the arm. More sophisticated collision checking and enforcing of joint limits were done on high-dimensional states.

We observed that inserting new spheres of radius of about 10cm allowed sufficient arm maneuvering without introducing too many unnecessary high-dimensional states. Also a tunnel radius of 10-20cm provides a good balance between the success rate of the tracking phase and the time needed for tracking a path at each iteration. Since we have a large number of high-dimensional states, we imposed time limits on both the adaptive planning phase and the tracking phase. The time limit we used for the adaptive planning phase was 180 seconds per iteration. If the limit was reached the adaptive planning failed and the algorithm terminated, reporting that no path from start to goal could be found in the given time limit. Due to the number of states inside the tunnel τ even with a small tunnel radius, the tracking search might take a long time to find a path through the tunnel or fail. Since we require the tracking search to complete before we begin a new iteration, it becomes impractical to wait long for tracking to fail before starting a new iteration. Thus, we limited the time for the tracking phase to 20 seconds, which allowed us to proceed to the next itration more quickly. Both the full 11D planner and the 11D/3D adaptive planner were limited to 600 seconds to produce a path.

The results we observed are summarized in Table I. As seen in the table, our adaptively dimensional planner was able to achieve much faster planning times than the full 11D planner and was able to successfully produce a solution in all 30 instances. The 11D planner, on the other hand, was much slower and was unable to find a solution within the allowed limit in 13 of the 30 instances. We observed an average speedup of x17.87. The minimal observed speedup was x1.12 on a very simple scenario that required only about 6 seconds to solve by both planners. In several cases, however, the adaptive planner was able to produce a solution within 5-10 seconds, while the 11D planner ran out of the allowed 10 minutes to produce a plan, giving us very high speedup values of over two orders of magnitude. On average, the sampling-based bi-directional RRT planner significantly outperformed both search-based planners. However, on the more cluttered environments, we observed that the adaptive planner was only marginally outperformed by the RRT planner, and in a few situations the RRT planner was actually slower than the adaptive planner.

We chose the task of manipulating sticks of varying

Algorithm	Suboptimality Bound	Time (secs)				# Iterations		# 11D Expands		# 3D Expands		Total Expands		Successful Plans
		mean	std dev	min	max	mean	max	mean	std dev	mean	std dev	mean	std dev	
11D	5.0	340.80	243.57	6.81	600.00	n/a		214K	159K	n/a		214K	159K	17 of 30
adaptive	5.0	19.07	16.65	5.35	55.44	1.30	3	10.2K	12.3K	67.1	30.79	10.2K	12.3K	30 of 30
RRT	n/a	4.15	6.25	0.02	25.69	n/a		n/a		n/a		n/a		600 of 600

TABLE I: Experimental results on 30 environments for 11D mobile manipulation planning (full 11D planner vs. adaptive planner vs. bi-directional RRT planner). The deterministic 11D and adaptive planners were run only once on each environment. RRT results are averaged over 20 runs on each of the 30 environments (600 runs total).

Algorithm	20cm stick					50cm stick					80cm stick				
	Time (sec.)				Success	Time (sec.)				Success	Time (sec.)				Success
	mean	std dev	min	max	rate	mean	std dev	min	max	rate	mean	std dev	min	max	rate
RRT (20 runs)	0.981	0.640	0.080	1.990	100%	33.885	36.474	0.320	130.270	100%	751.458	405.371	351.150	1176.66	20%
adaptive ($\epsilon = 5.0$) (1 run)	1.520	0.00	1.520	1.520	100%	3.540	0.00	3.540	3.540	100%	9.890	0.00	9.890	9.890	100%

TABLE II: Bi-directional RRT planner [11] vs. adaptive planner. The task was to manupulate a stick of varying length through a 40cm×50cm window similar to Fig. 6. RRT results are averaged over 20 runs with the same start and goal configurations. A time limit of 20min. was imposed on each run.

length trough a 40cm×50cm window as a basis for further comparison between our adaptive planner and the RRT planner. This task is challenging for sampling-based planners as it has a narrow solution space. The RRT planner needs to produce sufficently many valid samples within a narrow "tunnel", defined by the window, in order to successfully compute a feasible trajectory. From the results shown Table II we observe that increasing the length of the stick being manipulated causes a significant increase in the time required for RRT to produce a solution. On the other hand, our adaptive planner does not suffer such a significant performance decrease and it is able to significantly outperform the RRT planner on this scenario for large stick length values.

B. Experiments on PR2

We ran several real-world experiments on an actual PR2 robot using our adaptive planner. The experiments included tasks such as manipulating an 80cm stick through a window of size 40cm×50cm (Fig. 6), and reaching to and from shelfs of various heights (Fig. 7). All of the tasks required torso or base movement in order to complete successfully. The planner was able to successfully navigate from start to goal in all instances, and the planning times ranged from 4 to 20 seconds. Examples of the experiments are shown in the accompanying video.

VII. Discussion and Future Work

In the case of 11-DOF mobile manipulation planning we observed similar behavior of our adaptively dimensional planner as we did when applying it to other domains [18]. There are two cases in which our planner significantly outperforms full-dimensional weighted-A* planners: the case when the given scenario has no feasible solutions, and the case when the heuristic is misleading and leads the search into a wrong direction. Since our planner is working in a significantly smaller state-space, it is able to detect that no solution exists or that the search is going in the wrong direction by expanding significantly fewer states than the full-dimensional planner, despite the fact that it has to perform multiple iterations of searches. Our algorithm is also able to outperform sampling-based planners when the planning tasks require manipulating bulky objects through cluttered environments. A drawback of our algorithm is that its performance is highly dependent on the parameter values for the tunnel width and the size of newly inserted high-dimensional regions. These parameters affect the number of iterations performed by the algorithm and its running time. Narrow tunnels contain fewer states and can be searched quickly, but are more likely not to have a feasible path from start to goal. Introducing small high-dimensional regions keeps the size of S^{ad} small, and thus it can be searched quickly. However, the algorithm might have to grow them in subsequent iterations in order to meet the required solution suboptimality bound.

There are several key areas which we would like to explore further in order to improve the algorithm efficiency. Firstly, we would like to experiment with more informative heuristics, other than 3D Dijkstra heuristic for the end-effector, in attempting to speed up searching through the high-dimensional regions of G^{ad} and the tunnel τ. Another future goal is to investigate the possibility of re-using search information between subsequent iterations, in order to minimize the redundant state expansions, while still maintaining the completeness and suboptimality bound guarantees of our algorithm.

VIII. Conclusion

In this work we presented a heuristic search-based mobile manipulation planner, based on our recently-developed algorithm for planning with adaptive dimensionality. We have shown that by using dimensionality-reduction techniques our planner can effectively deal with the high dimensionality of the problem and demonstrate reasonable performance. Although our planner does not match the efficiency of sampling-based methods, our experimental results, coupled with strong guarantees on completeness and suboptimality bounds, make it an appealing alternative for solving mobile manipulation planning problems.

References

[1] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," in *Proceedings of Robotics: Science and Systems (RSS)*, 2008.

[2] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *International Journal of Robotics Research*, vol. 29, pp. 485–501, April 2010.

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

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

[5] ——, "Beam-stack search: Integrating backtracking with beam search," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005, pp. 90–98.

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

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

[8] 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, pp. 2902–2908.

[9] 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*, vol. 12, no. 4, pp. 566–580, 1996.

[10] S. LaValle and J. Kuffner, "Rapidly-exploring random trees progress and prospects," *Algorithmic and Computational Robotics New Directions*, pp. 293–308, 2001.

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

[12] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet, and J. Kuffner, "Manipulation planning with workspace goal regions," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 618–624.

[13] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research (IJRR)*, March 2011.

[14] Karaman and Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, June 2010.

[15] B. J. W. Waarsing, M. Nuttin, and H. V. Brussel, "Behavior-based mobile manipulation inspired by the human example," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003, pp. 268–273.

[16] ——, "Behaviour-based mobile manipulation: The opening of a door," in *International Workshop on Advances in Service Robotics (ASER)*, 2003, pp. 168–175.

[17] S. Chitta, B. Cohen, and M. Likhachev, "Planning for autonomous door opening with a mobile manipulator," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[18] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path planning with adaptive dimensionality," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2011.

[19] D. Tolani, A. Goswami, and N. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graphical Models*, vol. 62, pp. 353–388, Sep. 2000.