1-1-2013

Motion Primitives and Planning for Robots with Closed Chain Systems and Changing Topologies

Steven Robert Gray
University of Pennsylvania, stevegray954@gmail.com

Motion Primitives and Planning for Robots with Closed Chain Systems and Changing Topologies

Abstract
When operating in human environments, a robot should use predictable motions that allow humans to trust and anticipate its behavior. Heuristic search-based planning offers predictable motions and guarantees on completeness and sub-optimality of solutions. While search-based planning on motion primitive-based (lattice-based) graphs has been used extensively in navigation, application to high-dimensional state-spaces has, until recently, been thought impractical. This dissertation presents methods we have developed for applying these graphs to mobile manipulation, specifically for systems which contain closed chains. The formation of closed chains in tasks that involve contacts with the environment may reduce the number of available degrees-of-freedom but adds complexity in terms of constraints in the high-dimensional state-space. We exploit the dimensionality reduction inherent in closed kinematic chains to get efficient search-based planning.

Our planner handles changing topologies (switching between open and closed-chains) in a single plan, including what transitions to include and when to include them. Thus, we can leverage existing results for search-based planning for open chains, combining open and closed chain manipulation planning into one framework. Proofs regarding the framework are introduced for the application to graph-search and its theoretical guarantees of optimality. The dimensionality-reduction is done in a manner that enables finding optimal solutions to low-dimensional problems which map to correspondingly optimal full-dimensional solutions. We apply this framework to planning for opening and navigating through non-spring and spring-loaded doors using a Willow Garage PR2. The framework motivates our approaches to the Atlas humanoid robot from Boston Dynamics for both stationary manipulation and quasi-static walking, as a closed chain is formed when both feet are on the ground.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Mechanical Engineering & Applied Mechanics

First Advisor
Vijay Kumar

Second Advisor
Maxim Likhachev

Keywords
closed chain, motion planning

MOTION PRIMITIVES AND PLANNING FOR ROBOTS WITH CLOSED CHAIN SYSTEMS AND CHANGING TOPOLOGIES

Steven R. Gray

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2013

Acknowledgements

While at the University of Pennsylvania, I worked on many different projects which let me explore different aspects of robotics. To all those I worked with at Penn, Carnegie Mellon, Willow Garage, Lockheed Martin, and other partner institutions and organizations, thank you for helping me learn and experience so much.

I would like to thank my advisors, Vijay Kumar and Maxim Likhachev, for all of their support and advice. Vijay is a dedicated advisor who has let me choose and shape my involvement in the projects we worked on together. Max has always been there to help me with questions and think through solutions to those issues that inevitably crop up during implementation. I consider myself very fortunate to have had Vijay and Max as mentors. Additionally, I would like to thank Mark Yim, George Pappas, and Sachin Chitta for taking the time to serve on my dissertation committee. I would also like to thank all my friends and colleagues in GRASP for making the lab a great place to work and for making Philadelphia a great place to have spent these past six years.

I am immensely grateful to my family for their love and support. My parents and sister provided constant encouragement and motivation, always believing I could accomplish whatever I set out to do. My wife, Danielle, has been there for me every step of the way. She cheers me when I am down, lights a fire under my butt when I am listless, and I hope she knows how much I love her for it.

ABSTRACT

MOTION PRIMITIVES AND PLANNING FOR ROBOTS WITH CLOSED CHAIN SYSTEMS AND CHANGING TOPOLOGIES

Steven R. Gray

Vijay Kumar

Maxim Likhachev

When operating in human environments, a robot should use predictable motions that allow humans to trust and anticipate its behavior. Heuristic search-based planning offers predictable motions and guarantees on completeness and sub-optimality of solutions. While search-based planning on motion primitive-based (lattice-based) graphs has been used extensively in navigation, application to high-dimensional state-spaces has, until recently, been thought impractical. This dissertation presents methods we have developed for applying these graphs to mobile manipulation, specifically for systems which contain closed chains. The formation of closed chains in tasks that involve contacts with the environment may reduce the number of available degrees-of-freedom but adds complexity in terms of constraints in the high-dimensional state-space. We exploit the dimensionality reduction inherent in closed kinematic chains to get efficient search-based planning.

Our planner handles changing topologies (switching between open and closed-chains) in a single plan, including what transitions to include and when to include them. Thus, we can leverage existing results for search-based planning for open chains, combining open and closed chain manipulation planning into one framework. Proofs regarding the framework are introduced for the application to graph-search and its theoretical guarantees of optimality. The dimensionality-reduction is done in a manner that enables finding optimal solutions to low-dimensional problems which map to correspondingly optimal full-dimensional solutions. We apply this framework to planning for opening and navigating through non-spring and spring-loaded doors using a Willow Garage PR2. The framework motivates our approaches to the Atlas humanoid robot from Boston Dynamics for both stationary manipulation and quasi-static walking, as a closed chain is formed when both feet are on the ground.

Contents

Chapter 1

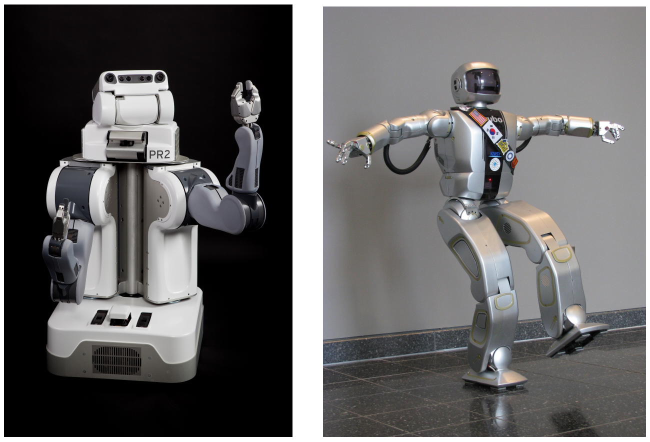
Introduction

Interacting with objects in the environment is becoming increasingly important in robotics. Robots are making inroads into human environments, from cleaning to patient care [37,45,93]. They are moving beyond the rigidity of assembly lines and fixed, repetitious motions [53,94]. However, to effectively interact with the world around them, including human environments, robots must be able to plan for situations in which multiple contacts are made with the world. Additionally, when operating in human environments, a robot should use predictable motions that allow humans to trust and anticipate its behavior. Heuristic search-based planning offers predictable motions and guarantees on completeness and sub-optimality of planned trajectories. While search-based planning on motion primitive-based (lattice-based) graphs has been used extensively in navigation, application to high-dimensional state-spaces has, until recently, been thought impractical. We present methods developed for applying these graphs to mobile manipulation, specifically for systems which contain closed chains. The formation of closed chains in tasks that involve contacts with the environment may reduce the number of available degrees-of-freedom but adds complexity in terms of constraints in the high-dimensional state-space. We exploit the dimensionality reduction inherent in closed kinematic chains to get efficient search-based planning.

1.1 Background

As the complexity of our robotic platforms increases, so does the need to plan in high-dimensional state-spaces. As little as a decade ago, low degree-of-freedom wheeled ground vehicles were dominant; now humanoid robots and mobile manipulation platforms abound. Two popular platforms, the Willow Garage PR2, with its holonomic wheeled base and dual arms (for a total of 20-degrees-of-freedom), and the humanoid HUBO, with 38-degrees-of-freedom, are shown in Figure 1.1.

Probabilistic sampling-based planning methods have come to the fore as a tractable means of searching high-dimensional spaces. However, they have a cost: these methods trade guarantees on solution completeness (with respect to geometric algorithms) and optimality (with respect to search-based algorithms) for speed, and must settle for probabilistic completeness guarantees. While RRT* and related works seek to recover path optimality, they may only do so in an asymptotic fashion; as the number of samples approaches infinity, they approach the optimal solution [56, 83]. Lately, there has been an upsurge in recovering optimal (with respect to discretization) solutions.



(a) **(b)**

Figure 1.1: The Willow Garage PR2 (a) with 20-degrees-of-freedom and the Korea Advanced Institute of Science and Technology (KAIST) HUBO (b) with 38-degrees-of-freedom are widely-used robotics platforms which require planners capable of handling many degrees-of-freedom.

Search-based planning is being applied to higher degree-of-freedom systems than ever before, as will be discussed in Chapter 2.

Planning for navigation, whether for field robots or robotic manipulators, typically involves planning through free space without additional constraints. Even in mobile manipulation literature, the common paradigm is to plan to get an end-effector near an object to manipulate, form the appropriate pre-grasp pose, approach the object to grasp, then move with the object attached, again planning without constraints [20, 22, 99, 111]. Other works have introduced constraints on manipulated object poses [7, 9]. However, there are instances when planning for mobile manipulation cannot be reduced to a series of open-chain planning problems. For instance, when interacting with an object constrained by or attached to the world in some way. Examples include opening doors and drawers, using levers and valves, and pushing objects along a track.

1.2 Motivation and Contributions

This thesis will demonstrate that search-based algorithms are applicable to systems with many degrees-of-freedom involving closed chains. The closed chains often arise in the form of contacts with the world, such as in mobile manipulation. Towards that end, we present a planning framework for search-based planning for mobile manipulators with changing system topologies; the systems contain closed chains, open chains, and transitions between the two. The planning framework is applied to the task of opening doors and is used to motivate our approach to bipedal locomotion. Door opening is an example of planning to manipulate an object along a constrained trajectory. Both tasks involve making and breaking closed chains.

First, Chapter 2 will provide an overview of the current state of the art in motion planning, planning for closed chains, and planning for mobile manipulation. Hierarchical planners and motion primitive-based planners will be covered. The algorithms involved span the gamut from deterministic, geometry-based planners to random sampling-based planners with probabilistic completeness guarantees to graph-based planners with their bounds on suboptimality of solutions. Additional literature pertaining to applications in later chapters will be discussed in those chapters.

In this thesis, we use search-based-planning algorithms. Chapter 3 begins with an overview of search-based planning, from Dijkstra's algorithm up to modern motion primitive-based approaches. This includes an overview of the A*, Weighted A*, and Anytime Repairing A* algorithms. We discuss lattice-based graphs, connected by motion primitives. Further, Chapter 3 provides descriptions of closed chains, systems in which they are likely to appear, and what constraints they impose.

Chapter 4 introduces our planning framework for handling open chains, closed chains, and transitions between them, all in a single planning instance, maintaining the completeness and optimality guarantees which would have been lost or weakened in a hierarchical planner. Theoretical guarantees are mentioned along with the necessary assumptions and conditions to apply the framework. A simple illustrative example is provided.

Chapter 5 covers application of our framework to the task of opening spring-loaded and non-spring-loaded doors using a mobile manipulator with a holonomic base. The door is constrained by its attachment to the world via revolute joint and may only move along a 1-D manifold, though it may move in either direction along that manifold. The robot must move its base to open the door and pass through the doorway. It may also switch between contacts with the door during planning and is allowed to contact the door with either arm, the base, or nothing at all.

Finally, Chapter 6 describes our approach to walking for a humanoid. We detail our control framework for the DARPA Robotics Challenge, specifically a balancing controller which has been extended to support quasi-static walking. In the double stance phase, the lower body forms a closed chain, while the single stance phase is an open chain. Planning for the humanoid involves abstractions for the complexity of kinematic chains from the pelvis to each foot. On that note, our planning framework inspires our work on walking, but the system is complex and the required assumptions and thus the guarantees of the framework do not apply directly. Our planning and control are able to successfully negotiate difficult terrain with hills and scattered obstacles.

Chapter 2

Literature Review

2.1 Background on Motion Planning

Motion planning difficulty depends on the dimensionality of the system and the constraints on the motion. Exact algorithms, which guarantee a solution when one exists and return failure when one does not, are limited to low-dimensional configuration spaces due to computational complexity. For instance, the most efficient exact algorithm has exponentially increasing complexity in dimensionality [17]. When considering the history of motion planning, we see that with exact methods infeasible, it was necessary to sacrifice exact completeness in order make the planning problem tractable. Discretization of dimension and configuration parameters was introduced. In general, algorithms based on an approximated cell decomposition of the free configuration-space [62] are resolution complete: they are complete for a given discretization size. In fact, narrow passageways smaller than the discretization size are guaranteed to be missed. Efficient heuristic algorithms have been developed based on the potential field approach [58]; these perform gradient descent on the potential field and may become trapped at local minima. Thus, the design of the potential function is crucial, but difficult for non-convex and high-dimensional spaces. All of the above approaches are applicable in practice to systems involving only a few variables, typically four or less.

Sampling-based planners, satisfying a weaker form of completeness but capable of handling high-dimensional configuration spaces, were introduced in the 1990s [57, 64]. These planners, probabilistic roadmaps (PRMs) and rapidly-exploring random trees (RRTs) guarantee probabilistic completeness; the planners sample randomly (or in a biased random fashion) from the configuration space. Such planners are not guaranteed to find a solution if it exists, often the case in the *narrow passage problem*, though variants have been introduced to handle such cases. There is also a drive to derandomize the sampling to improve coverage properties [63].

Lattice-based planners, used in this thesis, allow for planning as graph search. Such planners allow us to leverage results in graph search literature (A*, D* and their variants), previously applied to cell-decomposition methods. These methods have recently been shown to be applicable to high-dimensional state-spaces, as discussed in Section 2.4.

2.2 Planning for Generic Closed-Chain Systems

A number of planning methods exist specifically for closed kinematic chains. The works discussed below address planning for closed chain systems consisting of rigid links in 2-D (revolute and prismatic joints only) or 3-D (also includes ball joints).

Complete planners suffer from high computational complexity and difficult implementations. Some works, such as [103], are complete, but the path returned is not optimal by any metric. The cited work is valid for closed kinematic chains with spherical joints and will use at most $n - 2$ *accordion moves* to reach the desired configuration. The accordion moves are not optimal with respect to distance traveled nor any other metric. These works do not handle self-collisions or obstacles. An extension allowing point obstacles is limited to planar chains [67].

Early sampling-based methods for closed-chains were slow because the vast majority of generated samples did not satisfy the loop-closure constraints and so were rejected. A later sampling method, [112], applied to closed chains of a single cycle. It broke the closed chains into sub-chains, one of which used standard random sampling techniques, and the other which was populated using inverse kinematics to enforce the closure constraints. This method was applied to mobile manipulators and employed a two-stage PRM strategy. The first stage was generating a PRM for the manipulator at a single base location, and the second stage was replicating the PRM at different base locations and connecting them.

The work of [113] handled closed chains of any number of cycles. Each cycle was broken apart and gradient descent applied to minimize the sum of squares Euclidean distances between joints that should be collocated to satisfy the kinematic closure constraint. While samples could thusly be modified to satisfy the closure constraint, this was a time-consuming endeavor. In [101], the authors propose *planning with reachable distances*, precomputing and sampling directly from the subspace that satisfies the closure constraints. Closed chain systems are represented as a hierarchy of sub-chains; the corresponding reachable range of each can be computed using the triangle inequality. The method can be applied to most sampling-based planners, such as PRMs and RRTs. It has so far been applied to abstract chains in simulation.

The sampling-based works above have been applied to mobile manipulators, but do not allow for constraints on the motion of a manipulated object. The geometric methods have not been applied to mobile manipulation and cannot, for example, account for nonholonomic constraints of a robot base.

2.3 Application to Mobile Manipulators

As a central idea of this thesis is planning for closed-chain systems with specific application to mobile manipulation, we include an overview of other methods used in that field.

The idea of decomposing the motion of mobile manipulators into mobility and manipulation dates back to the first discussions of such systems in the 1980s. In [18], the task planning problem was formulated as a nonlinear optimization problem. Base and end-effector configurations were considered separately, and the optimization problem was solved for a sequence of manipulator and base configurations which minimized the cost function. Controls-based approaches have been used to drive a robot base in a manner that enables following specified end-effector trajectories [35]. Given a path for the end-effector, a feasible path is planned for the base such that the end-effector trajectory is always in the dexterous workspace. In this work, the dexterous workspace is always projected onto the ground plane, ignoring the height. Stability for the base and end-effector trajectory-following controllers is proven.

Finding the appropriate base position for manipulation tasks is not a trivial problem. Some work has applied probabilistic methods like rapidly-exploring random trees and probabilistic roadmaps to plan motions for mobile manipulators taking inverse kinematics and base position into account. For instance, [77] addresses the problem of motion planning along a specified end-effector path for a mobile manipulator with a nonholonomic wheeled base and kinematically redundant manipulator. For a given initial configuration, a path is assigned and a feasible solution generated using probabilistic methods. Redundant variables are chosen in advance; when sampling, values for these redundant variables are randomly generated, then the remaining variables are solved analytically. The redundant variables may also be generated by forward-integrating the equation of motion for the system using a random pseudo-velocity. Unlike our work, there are no optimality guarantees and only probabilistic completeness guarantees.

Probabilistic roadmaps (PRMs) have been used to solve multiple-query problems for mobile manipulators. Similar to RRTs, randomly sampled states are checked for feasibility, then connections between states are evaluated for feasibility. Unlike RRTs, a tree structure is not required; cycles are allowed and in fact add robustness. Work has gone to speeding this process up by only evaluating necessary connections between states (those required as the graph search is in progress), called Lazy PRM [11]. Lastly, it has been recognized that the probabilistic sampling is often unnecessary, leading to regularly-sampled versions, called LRMs [63].

In [104], solutions for dual-arm manipulation tasks are addressed for a fixed-base robot. First, the robot's reachability workspace is precomputed; it is represented by voxels in 6-D pose space and each voxel contains a probability of successfully answering an inverse kinematics (IK) query, i.e, solving for joint angles that produce the desired end-effector pose. Gradient descent is used to find a local maximum in the reachability space and combined with random sampling of free parameters. Then RRT-based motion planning algorithms are applied, interleaving finding IK solutions with searching for a collision-free trajectory. The work of [114] analyzes the manipulator reachability by discretizing the workspace with a regularly spaced spheres. On each sphere n-points are uniformly distributed, then frames are generated for each point on the sphere and serve as the tool center point for the inverse kinematics of the robot. Cross-correlation is used to decide the best base location for carrying out a predefined manipulator trajectory by mapping the trajectory to the closest frames on the spheres. In this work, the trajectory is required to be completely contained in the reachable workspace from a specific base location.

Berenson *et. al* introduce constrained bi-directional RRT for planning in configuration spaces with multiple constraints [10]. Pose constraints are handled by projecting sampled states onto configuration-space manifolds. The most common projection technique is the the Jacobian pseudo-inverse, in which the required workspace displacement to place the configuration back onto the manifold boundary is first calculated, then mapped into the joint space of the manipulator using the Jacobian inverse for square Jacobians or the Moore-Penrose pseudo-inverse for non-square Jacobians. The planner is able to handle moving heavy objects using sliding surfaces which support part of an object's weight. Dragging an object along a surface becomes an additional constraint manifold; if the object is not near a sliding surface but is too heavy to be lifted by the manipulator, the configuration is rejected. The advantage of our search-based method over this is that we may use the constraints to reduce the dimensionality of the state-space we are searching; the constraints enable a faster search.

CHOMP [92] is a trajectory optimization method that creates a naive initial trajectory from start to goal (unconstrained, it does not need to be a feasible trajectory), then runs a modified gradient descent on the cost function. The cost function typically has two components, one which is a cost for points along the arm which increases as they approach obstacles (this requires precomputation of a signed distance field) and one which enforces smoothness. Both of these are soft constraints; it is necessary to check the output of CHOMP for feasibility, i.e., being collision-free. The result of CHOMP is dependent on the initial trajectory given, as it may get caught in local minima. STOMP [54] is a similar approach which adds random noise to attempt to avoid local minima.

In contrast, [98] presents a trajectory optimization method which combines sequential convex optimization with a novel formulation of the collision avoidance constraint considering swept volumes. Rather than requiring precomputation of a signed distance field for the environment, this work requires either an approximate convex decomposition or a simplified mesh. Results compare favorably to CHOMP and STOMP. The work of Vernaza *et al.* focuses on identifying the low-dimensional Lagrangian structure of physical systems and applying this knowledge to aid in high-dimensional motion planning [105–107]. The algorithm learns and exploits the structure of holonomic motion planning problems using spectral analysis and iterative dynamic programming and is able to solve problems in higher dimensions than known methods for optimal motion planning. The quality of solutions found compares favorably to those obtained via sampling-based planning and smoothing.

RRT* is a recent development building upon RRT which converges to an optimal path as the number of samples tends to infinity. It adds a cost function which is used to calculate cost between vertices. When adding vertices to the tree, the edges are rearranged such that each vertex is reached using a minimum cost path from the tree root. The work of [83] extends RRT* to mobile manipulation tasks with many degrees-of-freedom by combining RRT* with the Ball Tree Algorithm. Other attempts to use probabilistic planners on costmaps include [6]. In this work, transitions to an increased cost state were allowed with a probability that depended on a thermal energy analog. When many subsequent expansions were rejected, the energy increased. This work does not provide guarantees on path optimality, but has a tendency to explore connected low-cost regions first and as such is appropriate for what the authors deem *cost space chasms*, narrow regions of low cost surrounded by higher cost, often of lower-dimensionality than the state-space and so unlikely to be found by naive sampling.

Hierarchical planning schemes have been proposed to reduce complexity by separating planning into smaller, simpler problems. Typically, a high-dimensional local planner is combined with a low-dimensional global planner. The typical benefit of a multi-level scheme is a significant reduction in planning time. Local planners have been implemented using various techniques, including reactive obstacle avoidance [102] and dynamic windows [15, 85]. While these types of planners can result in difficulties with suboptimality and mismatches between the local and global levels, our approach avoids these problems altogether by generating optimal plans in a low-dimensional space that maps to much higher-dimensional optimal solutions. Our representation has an additional advantage of incorporating a parameter to describe the contact state of the mobile manipulator and object being manipulated, allowing one plan to incorporate switching between open and closed- chain topologies while maintaining optimality.

2.4 Motion-Primitive Based Graph Planning

A key motivation of this thesis is to utilize lattice-based search for mobile manipulation. The advantage of search-based plans is bounded solution suboptimality, as well as determinism of solutions and completeness with respect to the discretization.

Searching motion primitive-based graphs has been applied to a variety of planning problems in robotics, including navigation problems [65]. Specifically, this work included planning dynamically-feasible maneuvers for vehicles at high speeds over large distances. Motion primitives searches have also been used for planning trajectories for UAVs [61,68].

Formulation of motion primitives is key in [88]. In order to allow backward searches such as D*, it is necessary to make sure the motion primitives can be connected forwards and backwards. Towards this end, they are constrained to begin and end on regularly discretized grid-cell centers. Motion primitives are pruned to generate near-minimal spanning action spaces by representing longer primitives as combinations of other primitives when possible. The work has also been extended to handle dynamics [89] by solving the corresponding two-point boundary value problem. The method has been shown to be valid for low-dimensional systems like wheeled vehicles, and has recently been extended to quadrotors [90].

Recently, heuristic search-based planners have been shown to be feasible for high-dimensional problems, specifically manipulation problems [23]. Planning times are lessened by combining informative but quickly computable heuristics (a Dijkstra search over the voxelized 3-D workspace), a small number of motion primitives, and an anytime, incremental graph search, Anytime Repairing A* (ARA*). An extension [25] incorporates motion primitives with variable dimensionality; these lower-dimensional primitives move a subset of the joints and are only used when near the goal end-effector position. Motion primitives which use inverse-kinematics to directly move from the current position to the goal are also used when near the goal. Additional constraints present in two-arm manipulation have been used to plan in a lower-dimensional graph [24]. A similar approach is used by [21], but their system lacks the ability to push and pull spring-loaded doors or make and break contact points with the doors. More recently, *experience graphs* have used results from previous searches have been incorporated to bootstrap solutions for new queries [86, 87].

Recent work has also been conducted on search-based planning with adaptive dimensionality [38, 39]. This uses the intuition that while planning in full-dimensional state-space is sometimes necessary, for large portions of the robot's workspace it is not. An adaptive-dimensional state-space and corresponding transition set is iteratively constructed that that consists mainly of low-dimensional states and transitions, using high-dimensional states only where necessary to ensure a feasible path. We incorporate this approach in our work on mobile manipulation with closed chains.

Search-based navigation planning for multiple robots has been addressed in the M*

algorithm [110]. Instead of planning for all robots in a configuration space containing the union of all robot configuration spaces, each robot is planned for separately. The configuration spaces are appended only if the robots are found to interact. In a process termed *subdimensional expansion*, the algorithm searches a lower-dimensional graph embedded in the full graph representing all the robots. Only when robots are found to interact are higher-dimensional state transitions back-propagated.

In this thesis, we consider a variety of closed-chain mobile manipulation tasks, including opening doors. This thesis represents the first work, to the author's knowledge, to incorporate transitions in system topology into search-based planning with motion primitives. The additional constraints from closed-chains are used to reduce system dimensionality in regions where the topology includes closed-chains, enabling efficient search. In contrast with the works above, the planning framework herein handles the transitions between open and closed chains in a single plan, including what transitions to include and when to include them. It does so while maintaining completeness and optimality guarantees over the entire plan, and the results share the determinism and repeatability inherent in graph-based search.

Chapter 3

Preliminaries

3.1 Graph Search

In this thesis, motion planning problems are represented by graph searches. Given a finite graph, we need an efficient way to search it for a solution path. We will now provide a brief review of graph search methods. We note that our planning framework with its dimensionality reduction is independent of the graph search chosen, though the specific guarantees of solution optimality do depend on the properties of the chosen search method.

Depending on the search space, the easiest method for constructing a graph is to discretize the state-space. For example, considering the 2-D search space in Figure 3.1, we can easily uniformly discretize the space, yielding a grid. Each discretized state corresponds to the center of a grid cell, and the edges represent motions from one cell

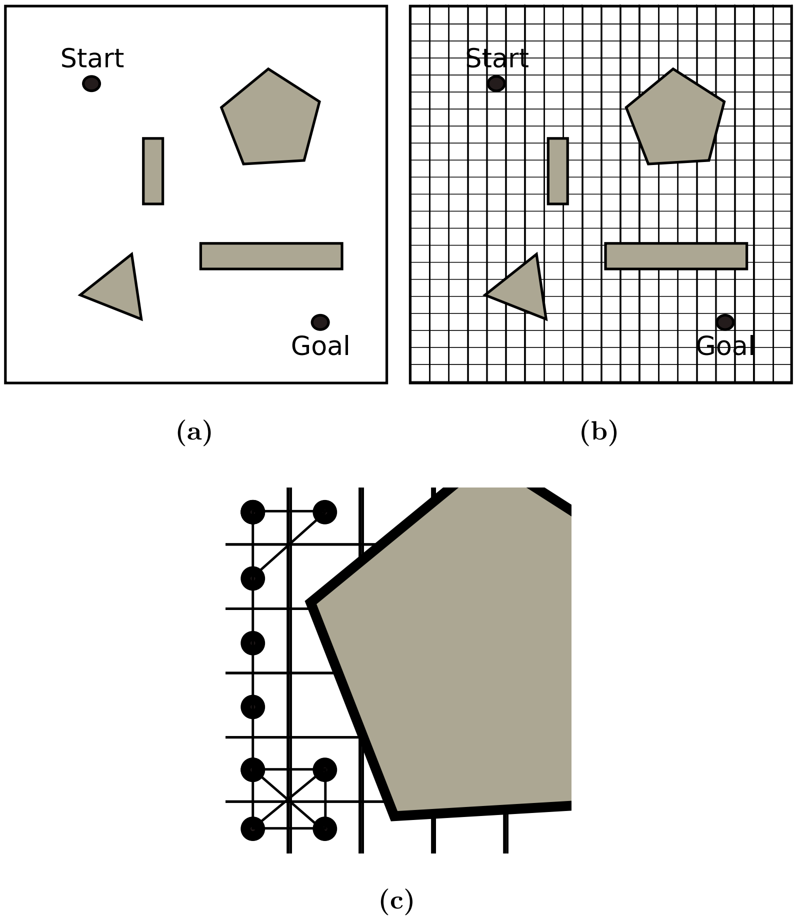


Figure 3.1: A 2-D state-space example. A configuration-space map (a) is uniformly discretized (b) and the underlying 8-connected grid is shown on a close-up view (c).

to another. Each edge has a weight which corresponds to the distance between cells. It is easy to envision using a 4- or 8-connected grid representation for the discretized space, as shown in the close-up in Figure 3.1(c).

Regardless of underlying representation, we discuss the planning problem as a graph, $G = [S, T]$, where S is the vertex set and T is the edge set. We define a set of transitions $T = \{a_{i,j} | s_i, s_j \in S\}$, where $a_{i,j}$ is a transition from state s_i to state s_j. Each transition is associated with a non-negative cost $c(a_{i,j})$, also written as $c(s_i, s_j)$. The objective of the planner is to find the least-cost path in G from start state s_{start} to goal state s_{goal}. Let $\pi(s_i, s_j)$ denote a path from s_i to s_j, and let $\pi^*(s_i, s_j)$ denote the least cost path. The path cost is the sum of the transitions along the path, $\sum_{i,j \in \pi} c(a_{i,j})$, denoted as $c(\pi(s_i, s_j))$.

3.1.1 Dijkstra Search

Dijkstra's algorithm [33] solves the single-source shortest path problem for a graph with non-negative edge weights. The g-value of a given state, $g(s)$, represents the current lowest cost path to reach that state. The search is initialized with $g(s_{start}) = 0$, and all other g-values set to infinity to show the search has not yet reached them. The start state is added to the $OPEN$ set. Then, at each iteration, the state with the smallest g-value is removed from the $OPEN$ set, and all the states connected to it (the possible *successor* states) have their g-values updated if the current state represents a lower cost way of reaching them. If so, the successor states are added to the $OPEN$ set. The process repeats until $g(s_{goal})$ is updated. The least cost path can be reconstructed by working backwards from the goal, always choosing the predicate state with the lowest g-value plus transition cost. Visualized on a 4 or 8-connected grid, Dijkstra's algorithm can be likened to a wavefront propagating outward from the start location.

Dijkstra's Algorithm

```
$g(s_{start}) = 0$, all other states set to $g(s) = \infty$
OPEN = $\{s_{start}\}$
**while** $g(s_{goal}) == \infty$ **do**
    remove $s$ with smallest $g(s)$ from OPEN
    **for** each successor $s'$ of $s$ **do**
        **if** $g(s') > g(s) + c(s, s')$ **then**
            $g(s') = g(s) + c(s, s')$
            Insert $s'$ in OPEN if not already present
```

3.1.2 A* Algorithm

A* search is a popular graph search algorithm [46], improving upon Dijkstra's algorithm by utilizing a heuristic to focus the search toward the goal. The search introduces the h- and f-values for a given state, where the h-value is the heuristic estimate of the cost to reach the goal and $f(s) = g(s) + h(s)$. The search is very similar to Dijkstra, except (1) the state removed from the $OPEN$ set each iteration is now the one with the lowest f-value and (2) the f-value is updated when the g-value is updated. The h-values must be an underestimate of the least cost path from the current state to the goal (*admissibility*) in order to ensure that the algorithm returns the optimal path. To ensure that no state is expanded more than once, the h-values must be *consistent*.

That is, for any two states where s' is a successor of s, $h(s) \leq c(s, s') + h(s')$.

A* Algorithm

```
$g(s_{start}) = 0$, all other states set to $g(s) = \infty$
OPEN = $\{s_{start}\}$
while $g(s_{goal}) == \infty$ do
    remove $s$ with smallest $f(s)$ from OPEN
    for each successor $s'$ of $s$ do
        if $g(s') > g(s) + c(s, s')$ then
            $g(s') = g(s) + c(s, s')$
            $f(s') = g(s') + h(s')$
            Insert $s'$ in OPEN if not already present
```

3.1.3 Weighted and Anytime Variants

The weighted version of A* works by biasing the sampling of new states toward the goal. For a given admissible heuristic function, multiplying the heuristic by a constant $\epsilon > 1$ and then performing the search as usual produces a solution with cost at most ϵ times the least cost solution [82]. In many domains, using the inflated heuristic greatly reduces the number of states expanded by the search before finding a solution.

While A* is able to find optimal plans, it can fail to find solutions when deliberation time is limited. Anytime planners, on the other hand, aim to find the best plan they can in the time allowed [28]. A (possibly) highly suboptimal plan is found and then improved and until either times runs out or the optimal solution is recovered.

In this thesis work, we have chosen to use an anytime heuristic search algorithm called Anytime Repairing A* (ARA*) [66]. The algorithm has control over the suboptimality bound of the solution it produces, which it uses to achieve the anytime property: it uses a loose bound to quickly find an initial solution, then tightens the bound progressively as time allows. Given enough time, it arrives at the optimal solution. ARA* reuses previous search efforts as it reduces ϵ and, thus, is more efficient than other anytime search methods. In comparison, a similar method called Anytime A* does not control ϵ directly (aside from setting the maximum ϵ during the first search) [115]. Instead, it continues to expand and re-expand states after the first solution is found by pruning states with f-values larger than the best cost solution so far.

3.2 Lattice State-Space

A state lattice, as described in [88], is a discretization of the configuration-space into a set of states and the connections between those states. Unlike the simple 8-connected grid representation of Figure 3.1, connections in the state lattice are required to be feasible motions of the system. Thus, any solution found while searching a lattice graph will also be feasible; the planner does not need to consider differential constraints directly.

The searches of this thesis are applied to regular lattices of states. Regularly sampled lattices provide translational invariance, in that a control or motion primitive connecting two states arranged in a certain way will also connect all other pairs of states arranged in the same way. Starting at a given location and applying the set of controls to all paths joining any discrete state state to its neighbors, one can form a roadmap or graph containing all trajectories possible given the discretization and choice of controls. The controls do not need to connect all adjacent states in the discretized space, but it is required that the states they connect are separated by multiples of the discretization value. These controls can be precomputed offline and stored as a canonical set of allowed motions. Barring obstacles, these motions encode the connectivity of the search space.

Motion primitives are defined as the smallest feasible motions that connect the discretized states in the graph. When planning a kinematic path, they are defined as small, kinematic displacements able to be tracked by the controller. Such is the case in [23–25]. When planning a kinodynamic path, the primitives correspond to known control inputs. Motion primitives can be designed by sampling the control space. Most such works attempt to do so in such a way as to result in good sampling in state-space in terms of discrepancy, dispersion or path diversity [13, 36, 44, 81]. In general, designing primitives this way is difficult due to the complexity of the relationship between the robot's control-space and state-space given the constraints upon the system. Finding control inputs that drive the system from one state to another can also be approached as solving a two-point boundary value problem for systems where that is possible [88, 89].

Similarly to how the state-space is discretized, we can think of discretizing the continuum of motions available to the system at a given state. This discrete set of motion primitives comprises the action space. There exists trade-off between including a wider variety of motion primitives and increased planning times. Smaller primitives may help the planner find paths in narrow passageways, while longer primitives may have faster planning times because fewer expansions are required. Ideally, the action space for each state in the lattice would contain a sufficient variety of motion primitives that every possible feasible path through the lattice could be constructed by combining sequences of these actions. Realistically, including a large number of primitives increases the branching factor of the graph to be searched and thus negatively affects the time required for each expansion, which is proportional to the total number of primitives. The best choice is often domain-dependent.

3.3 Closed Chains

A closed chain refers to a linkage whose kinematic structure contains one or more cycles. Compared to an open chain of the same number of links, closed chains have fewer degrees-of-freedom due to loop closure constraints. Namely, the product of all frame transformations around the chain must yield the identity tranformation.

$$\mathbf{T}_1 \cdots \mathbf{T}_n = \mathbf{I}$$

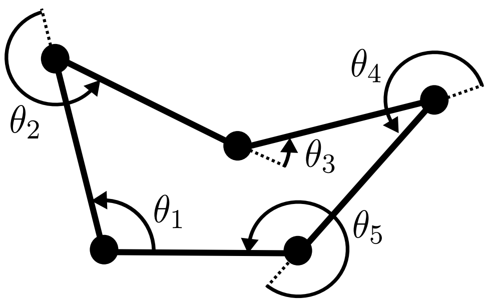
An open chain is considered kinematically redundant if it has more than the minimally required degrees-of-freedom to span the space. For planar manipulators, this value is 2; for spatial manipulators, it is 6. Consider a closed chain with one link held rigidly immobile. If the chain is kinematically redundant, the rest of the link will be able move; the valid motions of the closed chain comprise the self-motion manifold. Internal motions are those motions along the self-motion manifold and must satisfy

$$\mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = \mathbf{0}$$

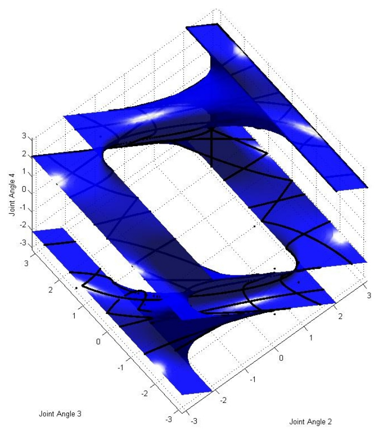
where $\mathbf{J}$ is the Jacobian and $\boldsymbol{\theta}$ the vector of joint angles.

Consider, for example, a linkage consisting of 5-revolute joints with parallel axes of revolution as shown in Figure 3.2(a). This planar 5-R linkage is kinematically redundant and has 2 degrees-of-freedom taking the loop closure constraint into account. If the linkage is equilateral, the configuration-space will look like Figure 3.2(b). For visualization, the configuration-space has been plotted as a function of three of the linkage joint angles, though it is actually a 2-D manifold. This manifold is the self-motion manifold; any trajectory along it will satisfy the loop closure constraint. Ignoring joint limits and self-collisions, the configuration-space can be endlessly replicated, forming a complicated structure without discontinuity but with numerous holes and thus many possible paths in different homotopy classes.

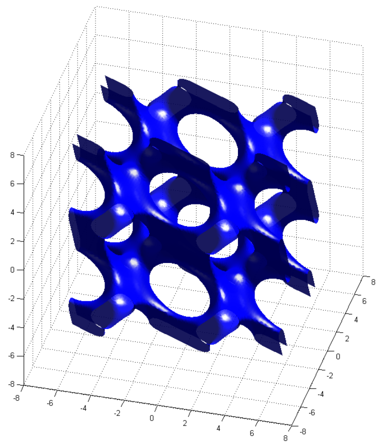
Because the space of valid configurations is a 2-D manifold, there is effectively no chance of randomly sampling a valid configuration if sampling for all 5 joint angles. There are methods that address this by breaking the closed chain into sub-chains and only sampling for one of them. Because there are 2 degrees-of-freedom in this chain, the active sub-chain will have 2 degrees-of-freedom and can use standard random sampling techniques, while the passive sub-chain with 3 joints and will be solved using inverse kinematics to enforce the closure constraints [112]. Alternatively, it is possible



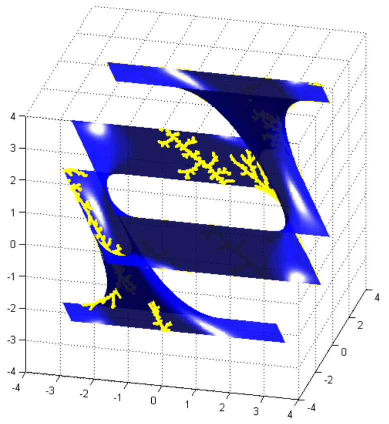
(a)



(b)



(c)



(d)

Figure 3.2: Example showing the configuration-space of a planar, equilateral 5-R linkage (a). The configuration-space is plotted as a function of the first three linkage joint angles. The robot has two degrees-of-freedom and so the configuration-space is a 2-D manifold, with black lines indicating stationary configurations (b). Ignoring joint limits and collisions, the configuration-space can be replicated endlessly (c). Standard planning methods, such as RRTs, can be applied to the manifold (d).

to sample directly in the space that satisfies the closure constraints [101] or to drive the full-dimensional samples to the constraint manifold [6, 113].

In screw theory, a *stationary configuration* occurs when any k joint screws belong to a screw system of order less than k where k must be less than the number of joints in the chain and greater than 1 [60]. In case of the 5-R example, stationary configurations occur when 3 or 4 of the joints are coplanar (though only 3 can be coplanar if the linkage is equilateral). Stationary configurations are shown with black lines in Figure 3.2(b). When 3 joints are coplanar, they have an instantaneous mobility of 1, and the other two joints are constrained so they only have one independent degree-of-freedom. Thus, if only those 2 joints were actuated, it would be impossible to navigate the control singularity. When 4 joints are coplanar, the remaining joint is transitorially inactive and cannot be used to control the robot through the singularity. Thus, when actuating closed chains, choosing which joints to actively control and which to make passive is very important.

Closed chains often arise in the form of contacts with the world, such as in mobile manipulation. When such systems makes and break contacts, they transition between open and closed chains. Bipedal walking is one example, in which the ground becomes the link which closes the chain formed by the pelvis and legs. When a bipedal robot is pushing against a wall with both hands, multiple closed chains arise; from each hand to each foot, hand to hand, and foot to foot. All these chains are closed by the external world.

Chapter 4

Planning Framework for Closed Chains and Systems with Changing Topologies

Motion primitive-based graph planning in high-dimensional systems is time consuming as planning times increase exponentially with increasing dimensionality. This is particularly a problem for mobile manipulation where the number of degrees-of-freedom is quite large. In addition, contacts between the robot and the environment result in the formation of *closed chain linkages*. A closed chain linkage is one whose kinematic structure forms a cycle. Such cycles introduce complex kinematic constraints, but can also be used to reduce the dimensionality of the planning problem. Examples of planning solely for closed chains are found in [67, 101, 103, 112, 113].

We propose a planning framework that handles systems with changing topologies, working with open chains, closed chains, and transitions between them. We propose abstracting away the complexity of closed chain systems to reduce the dimensionality of the planning problem in state-space regions where the closed chains exist, and give the conditions for completeness and optimality of solutions. For example, in the case of motion constrained to a plane, we may replace the manipulator with a two-degree-of-freedom linkage with two prismatic links but with a finite workspace and ignore the specifics of the manipulator in the abstraction. More generally, complex constraints associated with closed chains are replaced by abstractions that model the key aspects of the contact with the environment, removing unnecessary degrees-of-freedom and enabling switching between open and closed chain topologies within a single planner for mobile manipulation.

Several theoretical results provide the justification for the method and guarantees on optimality. The benefits of this planning methodology are verified through results and statistics from simulations involving a mobile platform with a planar arm moving an object along a plane. Applications to opening doors and walking are shown in Chapters 5 and 6, respectively.

4.1 Abstractions for Closed Chain Systems

Consider a mobile manipulation platform consisting of an n-degree-of-freedom manipulator atop a differential-drive base as shown in Figure 4.1. Let $\mathfrak{X} \subset SE(2)$

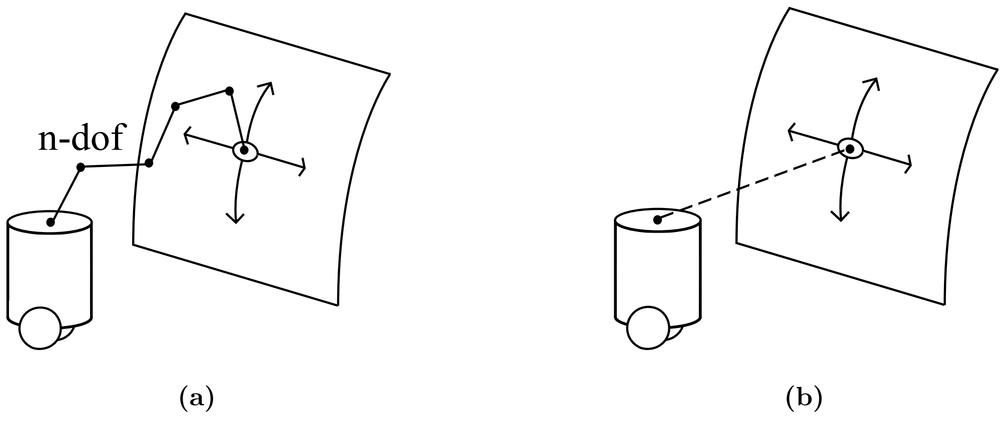


Figure 4.1: Example of an abstraction. Planning for an end-effector motion along a constrained manifold (a) may be simplified by planning only for the base and end-effector (b), then reconstructing the higher-dimensional path afterwards.

represent the set of configurations of the mobile base, $\boldsymbol{\mathcal{Y}} \subset S^1 \times S^1 \times \ldots \times S^1$ the set of manipulator arm configurations, and $\boldsymbol{\mathcal{Z}} \subset SE(3)$ the set of manipulated object configurations. $\boldsymbol{\mathcal{W}} = \{0, 1\}$ may be used to indicate whether or not the manipulator is in contact with and constrained by the object or the environment. The standard planning paradigm is to plan in $SE(2) \times \mathbb{R}^n$ (where we have replaced S^1 with $\mathbb{R}$), with appropriate constraints on end-effector motion. However, in many settings, mobile manipulation tasks may be encoded solely (but not uniquely) by the motion of the object being manipulated and the motion of the base.

Indeed, for a redundant manipulator, there may be infinite motions for the arm satisfying the end-effector motion. But it is clear that a sufficing plan can be found by restricting the search to $\boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{Z}}$ provided that, for every sufficing plan, feasible motions in $\boldsymbol{\mathcal{Y}}$ may be calculated, for example, using inverse kinematics methods. We note that this is feasible in general as long as the reachable arm configurations for a given pair $(\mathcal{X}, \mathcal{Z})$ define a path-connected set, ensuring the existence of transitions between consecutive inverse kinematics solutions (see Assumption 1 later in this chapter). For manipulators that do not satisfy this condition, and thus whose feasible configurations are in disconnected sets, we must limit ourselves to one such set. In the case of an $n \leq 6$ degree-of-freedom manipulator interacting with objects in $SE(3)$, replacing the manipulator in $\mathbb{R}^n$ with the object motion in $SE(3)$ does not reduce the dimensionality of the state-space. However, the proposed abstractions help in systems with $n > 6$ or when $n = 6$ but the object motion is only in $SE(2)$.

4.1.1 Planning Problem Formulation

We represent the full-dimensional planning problem as a graph, $G^f = [S^f, T^f]$, where S^f is the vertex set and T^f is the edge set. Let us define the full-dimensional (of dimensionality h) discretized finite state-space S^f as the 3-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, where $\mathcal{X} \in \boldsymbol{\mathfrak{X}}, \mathcal{Y} \in \boldsymbol{\mathfrak{Y}}, \mathcal{Z} \in \boldsymbol{\mathfrak{Z}}$. As in Figure 4.1, $\boldsymbol{\mathfrak{X}} \subset SE(2)$ is the set of configurations of the mobile base, $\boldsymbol{\mathfrak{Y}} \subset \mathbb{R}^n$ the set of manipulator arm configurations, and $\boldsymbol{\mathfrak{Z}} \subset SE(3)$ the set of manipulated object configurations. We emphasize that $\boldsymbol{\mathfrak{Y}}$ is finite, containing all valid manipulator configurations associated with positions chosen from $\boldsymbol{\mathfrak{X}}$ and $\boldsymbol{\mathfrak{Z}}$.

We define a set of transitions $T^f = \{a^f_{i,j} | s^f_i, s^f_j \in S^f\}$, where $a^f_{i,j}$ is a transition from state s^f_i to state s^f_j. Each transition is associated with a cost $c(a^f_{i,j})$, bounded from below by a positive constant δ. We have an edge-weighted graph G^f with vertex set S^f and edge set T^f. The objective of the planner is to find the least-cost path in G^f from start state s^f_S to goal state s^f_G. Let $\pi(s^f_i, s^f_j)$ denote a path from state i to state j, and let $\pi^*(s^f_i, s^f_j)$ denote the least cost path. The path cost is the sum of the transitions along the path, $\sum_{i,j\in\pi} c(a^f_{i,j})$, denoted as $c(\pi(s^f_i, s^f_j))$.

We note that the 3-tuple S^f is over-defined when an object is attached to the manipulator; $\{(\mathcal{X}, \mathcal{Y}) | \mathcal{X} \in \boldsymbol{\mathcal{X}}, \mathcal{Y} \in \boldsymbol{\mathcal{Y}}\}$ maps to a unique $\mathcal{Z} \in \boldsymbol{\mathcal{Z}}$ using the forward kinematics mapping f:

$$f(\mathcal{X}, \mathcal{Y}) = \mathcal{Z}.$$

In this thesis, we use a lattice-based graph representation to define the transitions between states, allowing motion planning problems to be formulated as graph searches, as discussed in Chapter 3. Lattices are well-suited to planning for constrained robotic systems because, unlike other graph-based representations such as n-connected grids, the feasibility requirement ensures that any solutions found using a lattice will also be feasible. We define a set of motion primitives as a set of precomputed kinodynamically feasible atomic actions. See Figure 4.2 for an example. We define a transition from the set T^f to be the result of a motion primitive applied to a state s^f. Let $\mathcal{A}^{\mathcal{X}}$ be the set of motion primitives for $\boldsymbol{\mathcal{X}} \subset SE(2)$ and $\mathcal{A}^{\mathcal{Z}}$ the set of motion primitives for $\boldsymbol{\mathcal{Z}} \subset SE(3)$. Let $\mathcal{A}^{\mathcal{Y}}$ be the set of corresponding motion primitives of $\boldsymbol{\mathcal{Y}} \subset \mathbb{R}^n$ that make the transition in $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Z}}$ feasible when the manipulator is grasping an object. $\mathcal{A}^{\mathcal{Y}}$ are defined as relative displacements and are dependent on the starting arm configuration $\mathcal{Y}$. When no object is being manipulated, we shall assume the manipulator moves

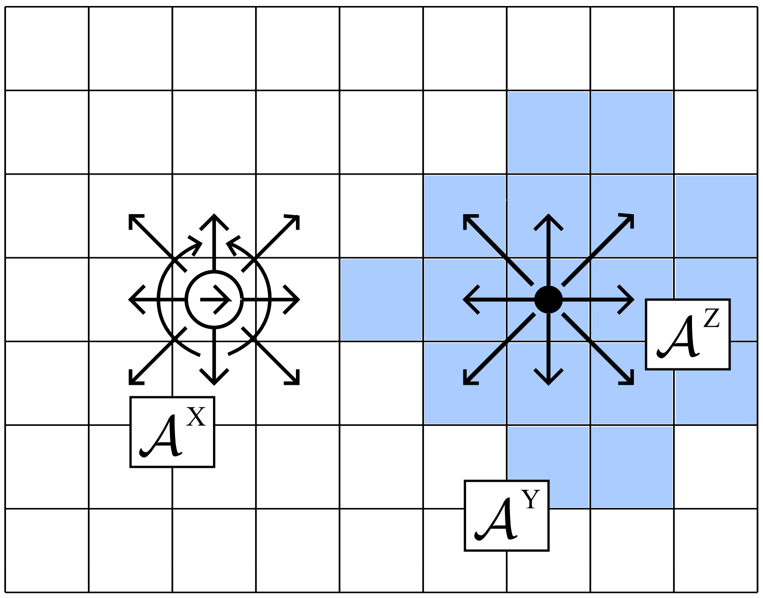

$\mathcal{A}^{\mathrm{X}}$
$\mathcal{A}^{\mathrm{Z}}$
$\mathcal{A}^{\mathrm{Y}}$

Figure 4.2: Example motion primitives for a mobile manipulator attached to an object. $\mathcal{A}^{\mathcal{X}}$ is the set primitives belonging to a mobile base with $\mathcal{X} = (x, y, \theta)$ constrained to move along an 8-connected grid or turn in place. $\mathcal{A}^{\mathcal{Z}}$ is the set belonging to an object with $\mathcal{Z} = (x, y)$, constrained to move on an 8-connected grid. In this case, $\mathcal{A}^{\mathcal{Y}}$ is the set of arm motions which keep the end-effector on the object during all transitions chosen from the feasible portion of $\mathcal{A}^{\mathcal{X}} \cup \mathcal{A}^{\mathcal{Z}}$. The shaded region represents reachable workspace of the manipulator.

freely. In the full-dimensional state-space, we define a motion primitive a^f as a 3-tuple member of the set $\mathcal{A}^f(\mathcal{Y}) = \{(a^{\mathcal{X}}, a^{\mathcal{Y}}, a^{\mathcal{Z}}) | a^{\mathcal{X}} \in \mathcal{A}^{\mathcal{X}}, a^{\mathcal{Z}} \in \mathcal{A}^{\mathcal{Z}}, a^{\mathcal{Y}}$ applied to $\mathcal{Y}$ enables $(a^{\mathcal{X}}, a^{\mathcal{Z}})\}$.

4.1.2 Reduced-Dimensional Graph

Let us also define a reduced-dimensional (of dimensionality l) discretized finite state-space S^l as the 2-tuple $(\mathcal{X}, \mathcal{Z})$. The crux of this work is that we may also represent the same mobile manipulation motion planning problem as a graph on the reduced-dimensional state-space $G^l = [S^l, T^l]$, where S^l is the vertex set and T^l is the edge set. S^l is a projection of S^f onto the lower-dimensional manifold. We define a many-to-one mapping $\gamma : S^f \to S^l$, in which $\gamma((\mathcal{X}, \mathcal{Y}, \mathcal{Z})) = (\mathcal{X}, \mathcal{Z})$, dropping the manipulator configuration $\mathcal{Y}$. We also define the inverse mapping $\gamma^{-1} : S^l \to S^f$, a one-to-many mapping. When an object is grasped by the manipulator,

$$\gamma^{-1}((\mathcal{X}, \mathcal{Z})) = \{(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) | \mathcal{Y} \in \boldsymbol{\mathcal{Y}}, f(\mathcal{X}, \mathcal{Y}) = \mathcal{Z}\}.$$

Otherwise, when no object is grasped (i.e., the manipulator forms an open chain),

$$\gamma^{-1}((\mathcal{X}, \mathcal{Z})) = \{(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) | \mathcal{Y} \in \boldsymbol{\mathcal{Y}}\}.$$

There is also a many-to-one mapping $\varphi : a^f \to a^l$, where a^l is a 2-tuple of the set

$$\begin{aligned} \mathcal{A}^l(\mathcal{Y}) = \big\{ &(a^{\mathcal{X}}, a^{\mathcal{Z}}) | a^{\mathcal{X}} \in \mathcal{A}^{\mathcal{X}}, a^{\mathcal{Z}} \in \mathcal{A}^{\mathcal{Z}}, \\ &\exists\, a^{\mathcal{Y}}, \mathcal{Y}\, s.t.\, (a^{\mathcal{X}}, a^{\mathcal{Y}}, a^{\mathcal{Z}}) \in \mathcal{A}^f(\mathcal{Y}) \big\} \end{aligned} \tag{4.1.1}$$

We require that edge costs be such that for every pair of states

$$c(\pi^*(s_i^f, s_j^f)) \geq c(\pi^*(\gamma(s_i^f), \gamma(s_j^f))) \tag{4.1.2}$$

The least cost path between any two states in the high-dimensional state-space is at least the cost of the least-cost path between their images in the low-dimensional state-space. The transition cost in the high-dimensional graph is

$$c(\pi(s_i^f, s_j^f)) = c_1(\pi(s_i^f, s_j^f)) + c_2(\pi(s_i^f, s_j^f)) \tag{4.1.3}$$

the sum of two terms that are not interrelated. The first term in Equation 4.1.3 is also the transition cost function of the lower-dimensional state-space; it is a function of only $\mathcal{X}$, $\mathcal{Z}$, $a^{\mathcal{X}}$, and $a^{\mathcal{Z}}$. The additive second term is a positive cost as a function of $\mathcal{Y}$ and $a^{\mathcal{Y}}$.

4.1.3 Algorithm

The overall algorithm is to construct and search the reduced-dimensional graph for a least-cost path from start to goal, then use that path to reconstruct one of the corresponding least-cost full-dimensional paths. This decouples the planning for the manipulator from the planning for the mobile platform. Reconstruction is done by traversing the path in the lower-dimensional space. Beginning with the first state, a $\mathcal{Y}$ is generated such that $s_1^f = (\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a valid configuration. Then the set of $\mathcal{A}^f$ is examined to find those transitions that produce the desired $(\mathcal{X}, \mathcal{Z})$ of the next state in the path. A $\mathcal{Y}$ is generated such that $s_2^f = (\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a valid configuration, and then any $a^{\mathcal{Y}}$ that produces s_2^f and satisfies Equation 4.1.1 is selected (either by planning or interpolation of inverse kinematics solutions).

When constructing the reduced-dimensional graph, we must verify the existence of some $a^{\mathcal{Y}}$ such that $(a^{\mathcal{X}}, a^{\mathcal{Y}}, a^{\mathcal{Z}}) \in \mathcal{A}^f(\mathcal{Y})$. This check can either be done using an inverse kinematics query at plan time, or done in advance as a precomputation (which is our chosen option). The precomputation is also used eliminate transitions with self-collisions. When constructing the graph during planning, we also must check transitions for collisions with the world. It is worth noting that both the high- and low-dimensional graphs include explicit transitions between grasping and not grasping an object.

```
ConstructFullDimPath($\pi^*(s^l_{start}, s^l_{goal})$)
  set $s^f_{start}$ to initial robot configuration
  **while** $s^l_n \neq s^l_{goal}$ **do**
      compute $\mathcal{Y}_n$ using IK seeded with $\mathcal{Y}_{n-1}$
      set $a^{\mathcal{Y}} = \mathcal{Y}_n - \mathcal{Y}_{n-1}$
      $n = n + 1$
      **if** transition $(a^{\mathcal{X}}, a^{\mathcal{Y}}, a^{\mathcal{Z}}) \in \mathcal{A}^f(\mathcal{Y})$, then **continue**
      **else** compute $a^{\mathcal{Y}}$ using arm planner
```

4.1.4 Theoretical Properties

We proceed to show that a graph search on G^l is sound, complete, and optimal. First, for convenience, let us define $\sigma : \pi(s^l_i, s^l_j) \rightarrow \pi(s^f_i, s^f_j)$, which maps a path in the lower-dimensional state-space to a set of corresponding paths in the higher-dimensional state-space.

Assumption 4.1.1. We assume that when the manipulator is connected to the object, the set $\boldsymbol{\mathcal{Y}}$ of feasible manipulator arm configurations for a given lower-dimensional state s_i^l occupies a path-connected set. For manipulators that do not satisfy this condition, and thus the feasible configurations are in disconnected sets, we limit ourselves to one such set.

That is, for any given $s_i^l = (\mathcal{X}_i, \mathcal{Z}_i)$, any corresponding feasible $\mathcal{Y}_i$ can be reached from any other feasible $\mathcal{Y}_j$. $\boldsymbol{\mathcal{Y}}$ forms a fully-connected set, though the connections are not required to be enumerated as part of $a^{\mathcal{Y}} \in \mathcal{A}^{\mathcal{Y}}$. When no object is grasped in the manipulator, $\boldsymbol{\mathcal{Y}}$ contains all possible manipulator configurations.

Theorem 4.1.2. *Soundness. Any path $\pi(s_i^l, s_j^l)$ in G^l can be executed in the full-dimensional space. That is, every $\pi(s_i^l, s_j^l)$ corresponds to at least one path $\pi(s_i^f, s_j^f)$ given by $\sigma(\pi(s_i^l, s_j^l))$.*

Proof. As our base case, we know the starting configuration may be mapped to the full-dimensional space. Assume the mapping σ exists and has produced $\pi(s_i^f, s_n^f)$, with $j > n > i$, terminating in $(\mathcal{X}_n, \mathcal{Y}_n, \mathcal{Z}_n)$. From the lower-dimensional path, we have $a_{n,n+1}^l = (a_n^{\mathcal{X}}, a_n^{\mathcal{Z}})$ and $s_{n+1}^l = (\mathcal{X}_{n+1}, \mathcal{Z}_{n+1})$. By Equation 4.1.1, there exists an $a^{\mathcal{Y}}$ such that for some starting configuration $(\mathcal{X}_n, \mathcal{Y}_j, \mathcal{Z}_n)$, there is $(a_n^{\mathcal{X}}, a^{\mathcal{Y}}, a_n^{\mathcal{Z}}) \in \mathcal{A}^f(\mathcal{Y}_j)$. However, $\mathcal{Y}_j$ may not be equal to $\mathcal{Y}_n$. Assumption 1 maintains that $\mathcal{Y}_j$ and $\mathcal{Y}_n$ are path connected, so we may transition from $\mathcal{Y}_n$ to $\mathcal{Y}_j$. Then, by definition, applying $(a_n^{\mathcal{X}}, a^{\mathcal{Y}}, a_n^{\mathcal{Z}})$ produces a valid state $s_{n+1}^f = (\mathcal{X}_{n+1}, \mathcal{Y}_{j+1}, \mathcal{Z}_{n+1})$. Thus, by induction, the entire corresponding path is given by $\sigma(\pi(s_i^l, s_j^l))$. $\square$

Theorem 4.1.3. *Completeness. If there exists a path $\pi(s_i^f, s_j^f)$ in the G^f, then there exists a corresponding path $\pi(s_i^l, s_j^l)$ in G^l.*

Proof. As our base case, we know the starting configuration may be mapped to the lower-dimensional space by dropping the $\mathcal{Y}$ component. Assume the mapping σ^{-1} exists and has produced $\pi(s_i^l, s_n^l)$, with $j > n > i$, terminating in $(\mathcal{X}_n, \mathcal{Z}_n)$. From the full-dimensional path, we have $a_{n,n+1}^f = (a_n^{\mathcal{X}}, a_n^{\mathcal{Y}}, a_n^{\mathcal{Z}})$ and $s_{n+1}^f = (\mathcal{X}_{n+1}, \mathcal{Y}_{n+1}, \mathcal{Z}_{n+1})$. The existence of $a_{n,n+1}^l = (a_n^{\mathcal{X}}, a_n^{\mathcal{Z}})$ is indicated by Equation 4.1.1, because $a_n^{\mathcal{Y}}$ exists. Applying $a_{n,n+1}^l$ to s_n^l results in the retrieval of the next state $s_{n+1}^l = (\mathcal{X}_{n+1}, \mathcal{Z}_{n+1})$. Thus, by induction, the entire corresponding path is given by $\sigma^{-1}(\pi(s_i^f, s_j^f))$. □

Theorem 4.1.4. *The cost of a least-cost path from start to goal in G^l is a lower bound on the cost of a least-cost path in G^f.*

$$c(\pi^*(s_S^l, s_G^l)) \leq c(\pi^*(s_S^f, s_G^f))$$

Proof. Theorem 2 established that the path $\pi_f^*(s_S^f, s_G^f)$ can be mapped onto the lower-dimensional state-space S^l. Given the restrictions on edge costs in Equation 4.1.2, the costs of any transition in G^l are bounded from above by the cost of any transition it maps to in G^f. Thus, with all transitions comprising the path bounded from above, the cost of a least-cost path from start to goal in G^l is a lower bound on the cost of a least-cost path in G^f. □

Theorem 4.1.5. *Optimality. If $c(\pi(s_i^f, s_j^f))$ does not depend on $\mathcal{Y}$ or $a^{\mathcal{Y}}$ (the second term in Equation 4.1.3 is zero), the mapping of the least-cost lower-dimensional path* *into the higher-dimensional state-space $\sigma(\pi^*(s^l_S, s^l_S))$, is also (one of) the optimal cost path(s) $c(\pi^*(s^f_i, s^f_j))$ in G^f.*

Proof. Theorem 4.1.2 established the mapping $\pi(s^f_i, s^f_j) = \sigma(\pi(s^l_i, s^l_j))$. Because transition costs are independent of $\mathcal{Y}$ and $a^{\mathcal{Y}}$, and because the full-dimensional states and transitions are mapped to the reduced-dimensional system by dropping only the $\mathcal{Y}$ and $a^{\mathcal{Y}}$ terms, the costs remain unchanged. Thus, the lowest cost path in G^l is one of multiple lowest cost paths in G^f due to the multiplicity of the mapping. □

By similar arguments, the paths in G^l that are ϵ-suboptimal (are of at most ϵ-times the cost of the least-cost path) are guaranteed to map to ϵ-suboptimal paths in G^f. This result is important when using ϵ-suboptimal searches like Anytime Repairing A*, used in the experiments in this chapter.

4.2 Proof of Concept

To demonstrate the benefits of our method, we test extensively in simulation on a system like that shown in Figures 4.1 and 4.2. We use a mobile base ($\boldsymbol{\mathcal{X}} \subset SE(2)$) with an n-degree-of-freedom planar arm ($\boldsymbol{\mathcal{Y}} \subset \mathbb{R}^n$) to move an object around the ground plane (the object has no notion of directionality, so $\boldsymbol{\mathcal{Z}} \subset \mathbb{R}^2$). The configuration space has been inflated so the robot and object can be represented as points. The manipulator may attach and detach from the object, switching between open and closed chains, so $\boldsymbol{\mathcal{W}} = \{0, 1\}$. The object can be thought of as a cylinder with omnidirectional wheels; the planar arms make contact with the cylinder at a height greater than the height of any world obstacles. Thus, obstacles can collide with the mobile base and the object being moved, but not with the arms. When the arm makes contact with the cylinder, we assume it is rigidly attached.

Any state in the full-dimensional state-space is given by

$$s^f = (\underbrace{x_r, y_r, \theta_r}_{\mathcal{X}}, \underbrace{\theta_1, \ldots, \theta_n}_{\mathcal{Y}}, \underbrace{x_o, y_o}_{\mathcal{Z}}, \underbrace{m}_{\mathcal{W}})$$

where $(x_r, y_r, \theta_r) \in \mathfrak{X}$ is the mobile base pose, $(\theta_1, \ldots, \theta_n)$ is the joint angles of the arm, $(x_o, y_o) \in \mathfrak{Z}$ is the cylinder location, and m is a binary value indicating if the object is attached to the manipulator. A state in the reduced-dimensional state-space is given by

$$s^l = (\underbrace{x_r, y_r, \theta_r}_{\mathcal{X}}, \underbrace{x_o, y_o}_{\mathcal{Z}}, \underbrace{m}_{\mathcal{W}})$$

4.2.1 Implementation

The goal of the planner is to get the object to a desired location on the 2-D grid. The robot must navigate to the object, attach it to the manipulator, then move it along the plane to the goal while avoiding obstacles. There is no fixed goal for the location of the robot base. The heuristic function used to guide the search is the distance between the robot and object plus the distance between the object location and the goal. When the robot is connected to the object, only the latter is used. Both distances are calculated for the entire map during precomputation using 2-D Dijkstra searches.

The cost function is:

$$c(a_{i,j}^{l}) = (c_{costmap}(a^{\mathcal{X}}, \mathcal{X}) + 1)(c_{movement}(a^{\mathcal{X}}) + c_{obj}(a^{\mathcal{Z}}, \mathcal{Z}))$$

where $c_{costmap}(a^{\mathcal{X}}, \mathcal{X})$ represents the maximum cost cell traversed during the transition, $c_{movement}(a^{\mathcal{X}})$ the cost associated with moving the robot, and $c_{obj}(a^{\mathcal{Z}}, \mathcal{Z})$ the cost associated with moving the object. The cost is independent of manipulator motion, satisfying Theorem 4.1.5. The units for the cost functions are seconds; the movement cost for forward or backwards motion is the distance divided by nominal velocity of the robot and the cost for turning in place is the angular distance divided by the nominal angular velocity. The cost for moving the object is similarly a distance divided by nominal velocity for the object (one could think of it as the speed at which the object may be moved without tipping). The costmap is unitless with a value of 0 for unoccupied space and 254 for the obstacles themselves.

A valid configuration of the arm, $\mathcal{Y}$, when not connected to the object is any configuration not in self-collision. A valid $\mathcal{Y}$ when connected to the object must satisfy the $(\mathcal{X}, \mathcal{Z})$ pair. Such pairs are constrained such that the object is at least one arm link length distant from the base, but no more than the total length of the arm. The arm is assumed to be above the height of the obstacles and so cannot collide with them. This, coupled with the distance constraint, satisfies the path connectivity requirement of $\boldsymbol{\mathcal{Y}}$.

The reduced-dimensional lattice is constructed using 12 motion primitives, of which 8 are for moving the object in a 8-connected grid, 2 for turning base in place, and 2 for forward and backwards movement. The robot arm is allowed to connect to the object, but not to disconnect from it. ARA* is first run on the reduced-dimensional state-space representation, initially with suboptimality bound $\epsilon = 5.0$ and continued until $\epsilon = 1.0$. Full-dimensional paths are generated by populating the arm joint angles using inverse kinematics (using an iterative method, seeded with the previous state's joint angles). If the interpolation fails to connect two solutions without self-collision, a random arm configuration is generated, checked for collision, then used as the seed for the inverse kinematics call. This method succeeded in generating a full-dimensional path in all trials.

4.2.2 Results

We tested the graph planner with closed chain abstractions on 100 randomly-generated maps of size 100 by 100 cells, 10 cm on a side. 50 of these were pseudo-outdoor environments (random circular obstacles) and 50 were pseudo-indoor environments (grid obstacles). Robots with planar arms of 3 or 10-links were used; the 3-link arms had link lengths of 10 cm, while the 10-link arms had link lengths of 4 cm. The graph planner results were compared against those found by a sampling-based planner, RRT [57,64]. The RRT was implemented as two successive searches; the first (S1) was to bring the robot end effector into contact with the object and the second (S2) to move the robot end effector, now with object attached, to the goal location. The RRT was unidirectional, with the root at the robot start location. At each iteration, the

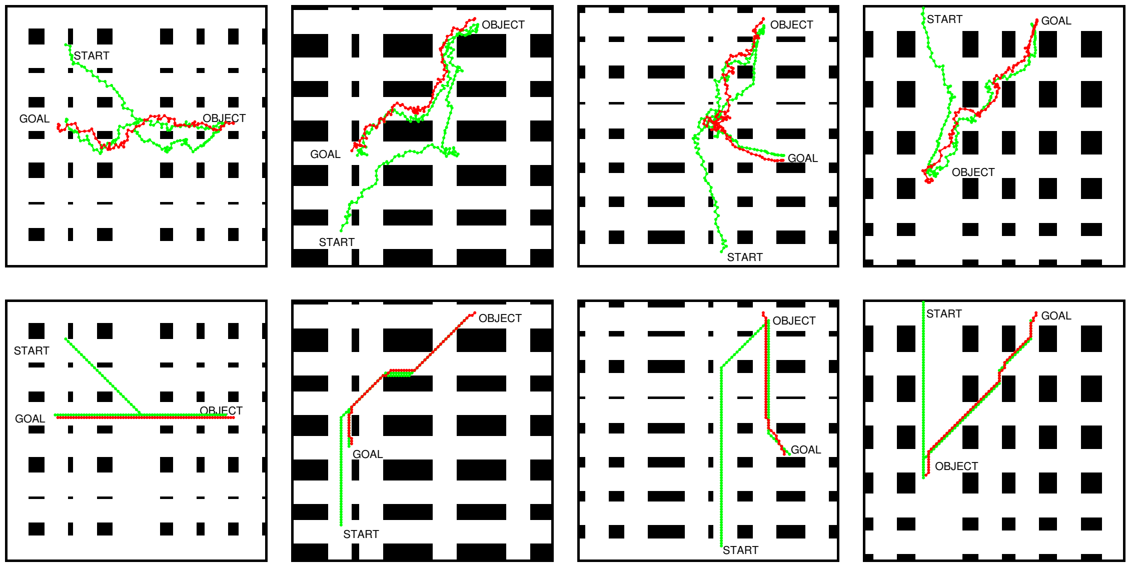


Figure 4.3: Comparison of indoor plans between RRT (top row) and graph planner (bottom row). Environments are 100 by 100 cells with 10 cm cell sides. The robot has a 10-link arm, with each link being 4 cm in length. The robot must approach an object, pick it up with the manipulator, and bring it to the goal. The green indicates the robot base trajectory and the red indicates the trajectory of the object when attached to the manipulator. The arms are not shown.

probability of sampling from the goal region was 0.02. Goal sampling was accomplished by sampling within an annulus determined by the distance constraints of the object or object goal, then using inverse kinematics to get feasible arm joint angles. Any sample, after being checked for collision (between the robot base or object and obstacles) and self-collision, was connected to the nearest state in the tree provided the interpolation between the two was also collision-free.

For the graph search, planning times and number of states expanded are shown for

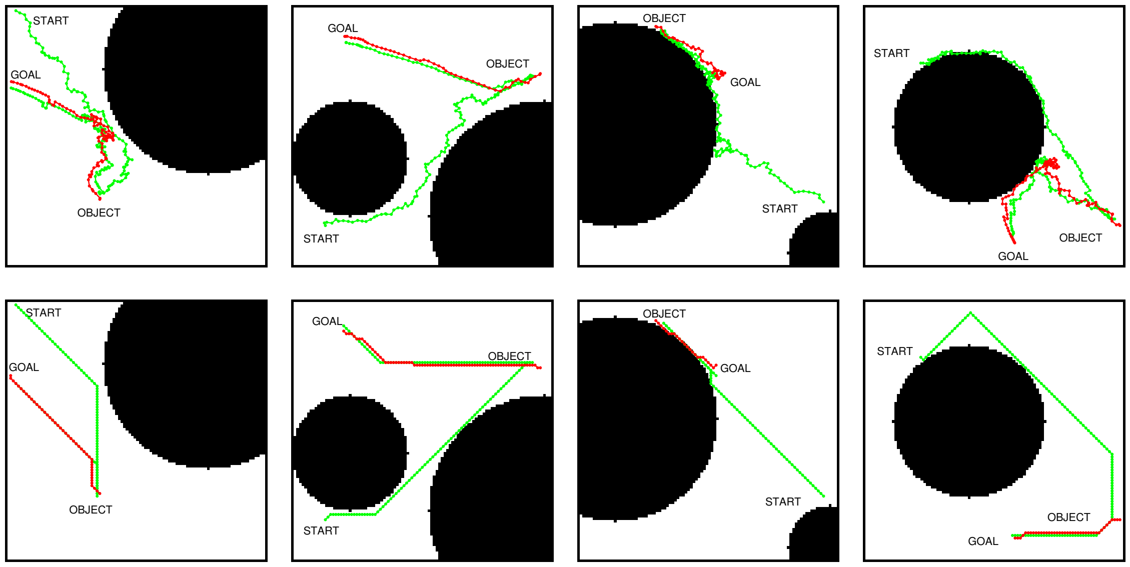


Figure 4.4: Comparison of outdoor plans between RRT (top row) and graph planner (bottom row). Environments are 100 by 100 cells with 10 cm cell sides. The robot has a 10-link arm, with each link being 4 cm in length. The robot must approach an object, pick it up with the manipulator, and bring it to the goal. The green indicates the robot base trajectory and the red indicates the trajectory of the object when attached to the manipulator. The arms are not shown.

the initial plan with $\epsilon = 5.0$ and the final plan with $\epsilon = 1.0$. The reduced-dimensional planning time is given by LD, and the full-dimensional path reconstruction from the reduced-dimensional path is listed under FD. Reconstruction was only done for the $\epsilon = 1.0$ paths. Results for indoor and outdoor environments are shown in Tables 4.1 and 4.2, respectively. Sample runs (with the arms not pictured for clarity) are shown in Figures 4.3 and 4.4.

It is worth noting the RRT failed to plan in under 30 seconds for the 10-link chain

	Graph Search				
Arm DOFs	Subopt. Bound ϵ	Planning Times (s) LD	Planning Times (s) FD	Expansions	Success
3	5	$< 0.01 \pm < 0.01$	N/A	287 ± 139	50/50
	1	2.30 ± 2.45	$< 0.01 \pm < 0.01$	$3.2 \times 10^5 \pm 3.0 \times 10^5$	
10	5	$< 0.01 \pm < 0.01$	N/A	288 ± 164	50/50
	1	3.48 ± 3.19	0.023 ± 0.012	$4.78 \times 10^5 \pm 4.17 \times 10^5$	

	RRT		
Arm DOFs	Planning Times (s) S1	Planning Times (s) S2	Success
3	0.13 ± 0.12	0.11 ± 0.09	50/50
10	2.81 ± 3.28	4.24 ± 6.08	47/50

Table 4.1: Graph planner and RRT planner comparison for simulated indoor environments.

	Graph Search				
Arm DOFs	Subopt. Bound ϵ	Planning Times (s) LD	Planning Times (s) FD	Expansions	Success
3	5	$< 0.01 \pm < 0.01$	N/A	280 ± 154	50/50
	1	1.39 ± 1.63	$< 0.01 \pm < 0.01$	$2.36 \times 10^5 \pm 2.71 \times 10^5$	
10	5	$< 0.01 \pm < 0.01$	N/A	253 ± 139	50/50
	1	2.57 ± 2.50	0.024 ± 0.013	$3.39 \times 10^5 \pm 3.19 \times 10^5$	

	RRT		
Arm DOFs	Planning Times (s) S1	Planning Times (s) S2	Success
3	0.20 ± 0.44	0.14 ± 0.27	50/50
10	1.87 ± 1.95	1.29 ± 1.03	46/50

Table 4.2: Graph planner and RRT planner comparison for simulated outdoor environments.

in 4 outdoor trial environments, and 3 indoor trial environments. These environments featured narrow passageways that a vanilla RRT is ill-equipped to handle (though variants such as [96] help handle such regions). The outdoor environments are shown in Figure 4.5. The graph search was successful on all environments in the same time limit.

Path length comparisons were done on a specific indoor environment, with the

	Graph Search		RRT	
Arm DOFs	Base Path Length	Object Path Length	Base Path Length	Object Path Length
3	8.60 ± 0.53	4.86 ± 0.41	19.88± 4.07	13.61± 3.53
10	8.46 ± 0.53	4.81 ± 0.40	18.53± 2.89	12.33± 2.50

Table 4.3: Graph planner and RRT planner comparison of path lengths for the robot base in perturbations of a simulated indoor environment.

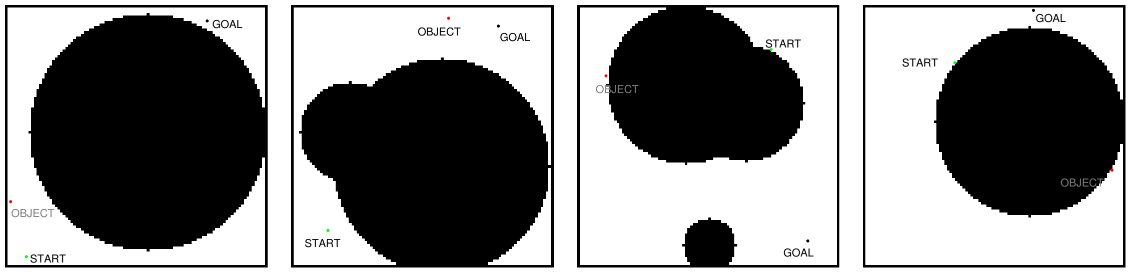


Figure 4.5: Environments in which the RRT fails to find a solution in under 30 seconds, but the graph planner succeeds. These environments are characterized by narrow passageways the robot must traverse.

start and goal perturbed slightly each time. Results from these runs are given in Table 4.3 and highlight the benefits of the graph search, determinism and repeatability. Similar problems will have consistent, similar, optimal solutions. The graph search path lengths for the base and the object are less than half of those for the RRT, and the corresponding standard deviations are less than one-fifth that of their RRT counterparts.

4.2.3 Discussion

We have proposed a methodology for reducing the dimensionality of search-based planning problems for mobile manipulators by creating lower-dimensional abstractions for state-space regions which contain closed chains. The methodology is able to handle transitions between open and closed chains in a single plan and, in fact, the planner inserts those transitions as necessary. For mobile manipulators, the central idea is to only retain the configuration space of the mobile base and the objects in the environment without explicitly modeling the configuration of the arm or the constraints associated with closed-chain systems that are formed when the arm contacts objects in the environment. The mathematical formulation for our approach allows us to prove optimality and completeness under reasonable assumptions. While we have illustrated the results in a simple scenario, the framework and approach are applicable to a wide variety of mobile manipulation tasks involving contact with the world.

Chapter 5

Case Study: Door Opening

In this chapter, we apply, explore applications, study the scalability, and validate the motion planning framework from Chapter 4 on the problem of opening doors. Opening doors is necessary in order to enhance and expand the set of tasks an autonomous robot can perform indoors. The majority of commercial doors are equipped with automatic mechanisms to ensure closure. These types of doors are typically called *spring-loaded* doors, whereas doors that do not close automatically are known as *non-spring-loaded*. Autonomously planning for opening both spring- and non-spring-loaded doors is essential to provide the functionality required of a useful indoor robot. One must tackle several problems in order to build an integrated door opening implementation for a mobile manipulation platform, such as detecting the type of door and the location of the handle, building a kinematic model of the door, and coordinating the arm and base of the robot to open the door with respect to space constraints in the immediately surrounding area. Our focus is on the last issue, *i.e.,* planning for and coordinating the motion of the arm and the base to approach, open, and pass through doors.

Although door opening in indoor environments has been widely addressed in recent work for mobile manipulation systems, [21, 52, 59, 74, 76], opening and moving through doors is still a challenging problem. Doors vary with respect to size, shape, space constraints, and handles; therefore, hard-coded and precomputed motions designed to open doors can easily fail when designing a robust system. Reactive approaches and low-level controllers may fail to consider obstacles and may need to be modified to handle doors with different parameters. Opening and navigating through doors, especially spring-loaded doors, requires making and breaking contacts with the door, making it well-suited to our planning framework. For spring-loaded doors, the robot must maintain contact with the door and actively counteract the spring to keep it from closing. For doors in cluttered or confined environments, it is often necessary to switch the side of the door the robot contacts. This thesis contains, to the best of the authors' knowledge, the first planning framework that handles non-spring and spring-loaded doors, functions in cluttered or confined workspaces, and plans the approach to the door, pushing or pulling it open, and passing through. The plan is generated in a unified search space, including transitions between different robot-door contacts (for example, base against the door as well as gripper on the door handle), finding a least-cost solution for traversing doors. This is important because it allows the planner to decide the best location and time to transition from opening the door

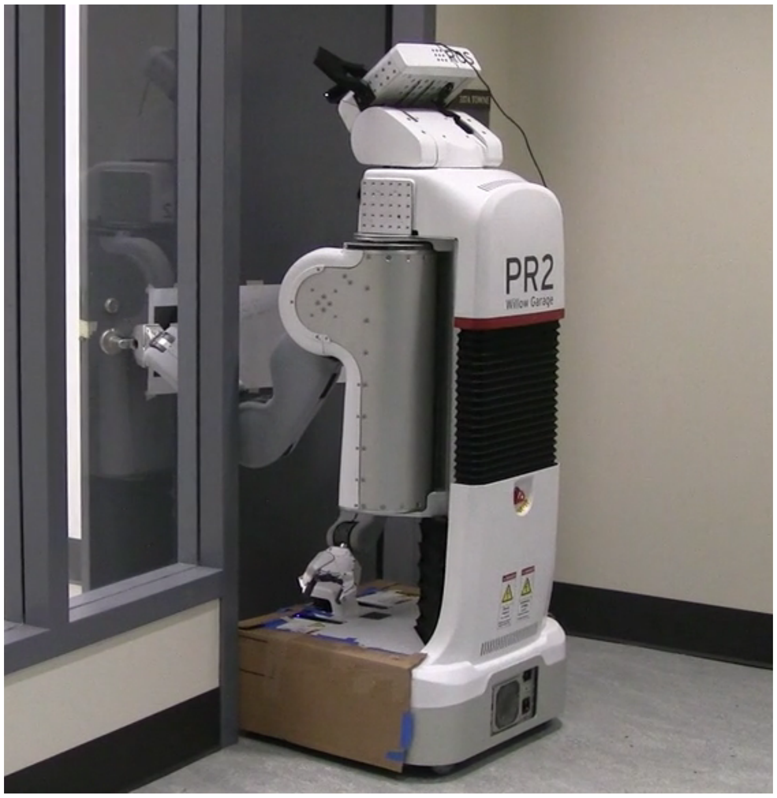


Figure 5.1: The PR2 pushing open a spring-loaded door. Once the base is in contact with the door, the arm can let go of the door and use the base to keep pushing the door open as it moves through.

to moving through it. In contrast, disjoint approaches like hierarchical planners will specify but not modify the transitions.

Our approach to motion planning starts with using a low-dimensional, graph-based representation of the problem in order to plan a door-opening procedure quickly and reliably. This provides several advantages including the ability to quickly plan for opening various doors and account for walls or other obstacles in the surrounding environment. If necessary, contact must be transferred very carefully between the arms and the base of the robot to open and maintain the position of the door throughout execution of the opening action. Our planning algorithm can take into account which (if any) part of the robot is currently holding the door open and allows for transitions between arms as well as using the base to push against the door. We note that, particularly for pulling spring-loaded doors, it is very difficult to open the door and maneuver such that the base is in position to hold the door open as it passes through the door frame. Our planner handles this difficult situation.

A robotics platform for use in door opening is greatly aided by having some method for manipulating the door from both sides. For the purposes of this work, we assume a dual-arm mobile manipulator is used. Our approach is validated by an extensive set of experiments performed using the PR2 robot, a platform used extensively for navigating and acting within indoor environments [70, 72]. Our experiments involved multiple tests for opening a variety of doors (pulling and pushing both spring- and non-spring-loaded doors).

5.1 Related Works

Robotic door opening has received a fair amount of attention in recent years. Within the overall task of robots opening doors, research can involve visual identification of doors and door handles [3, 5, 59, 80] or the physical action of opening the door. We simplify door identification since our contribution relates to the planned robotic motion required for door opening; door detection is outside the scope of this work.

We chose a simple visual identification system based on the ARToolKit [34], but any other can be used.

Recent work has seen a number of robotic platforms addressing door opening. Early experiments [74, 76] have led to a number of systems designed for this task. However, as we mentioned in [21], many of these systems have not completely solved the door opening problem. Some do not pull open doors [59, 72, 79, 80]. Others such as [97] use impedance control to open doors while learning the kinematic model, but may hit obstacles and do not move the robot base.

Purely reactive approaches such as [84, 95] will not work in complex environments and for multiple contacts. Planning algorithms provide a way to take into account factors that may not have been considered when designing a completely precomputed action or a reactive controller. There are also planning approaches that do not include motion of the robot base while opening doors. [30] plans manipulation motions for the opening of cabinet doors, allowing switching between different caging grasps, but the base of the robot remains stationary. Task space regions can accommodate pose constraints [9], but have not been used to simultaneously plan arm and base trajectories for door opening.

There have been other recent approaches to door-opening systems. However, none of these combine all of the following: switching contacts to allow end effectors or the body of the robot to brace the door; the ability to handle spring-loaded doors; and combining the approach to the door, including selecting an initial contact with the door, and opening/passing through the door as a single plan. A number of trajectory planners which take into account external wrenches on the end-effector have been developed both for wheeled platforms [55,91] and humanoid platforms [4,32]. However, these methods only allow pushing doors and cannot switch contact locations. In [109], the authors create a behavior-based system able to push and pull doors open, but do not deal with spring-loaded doors and obstacles. In [2], the authors estimate door parameters and pull open a door with a modular re-configurable robot featuring passive and active joints, but cannot pass through spring-loaded doors. In [27], Dalibard *et al.* introduce random sampling planning algorithms for humanoid robots to move through a doorway while also opening and closing the door. They have demonstrated results using both arms of the robot to move through the door and avoid obstacles, but are unable to pass through spring-loaded doors. In [52], Jain and Kemp extend previous work in which a robot could successfully open doors and drawers to include motion of the robot base. Their work does not include switching contact locations and would not allow the robot to brace open and pass through a spring-loaded door.

We build on the prior work of Chitta, Cohen, and Likhachev [21], in which collision-free trajectories were generated for opening non-spring-loaded doors. The previous system was limited to a single contact, the end-effector upon the door handle. We build upon this by adding transitions between robot-door contacts, including planning for the initial contact with the door. Pushing and pulling spring-loaded doors are handled by incorporating additional constraints on maintaining contact with the door.

State feasibility is determined by checking against a precomputed map of the force-workspace. In contrast to previous work, the entire plan from approach, to opening, to moving through doors is computed in a single search. This allows a least-cost solution to be found that ensures the feasibility of contact transitions. Results have been shown at [40–42].

5.2 Motion Planning

We solve the planning problem for approaching, pushing and pulling open both non-spring and spring-loaded doors, and moving through them. The approach is most beneficial for doors that cannot be fully opened while the base remains stationary, but is also useful for other doors in constrained spaces. When attached to the door, we constrain the motion of the manipulator to a 1-D manifold traced by the path of door handle.

The door-opening problem is to find a configuration path such that:

1. The robot passes through the door
2. The robot avoids self-collision and collision with obstacles
3. The path is feasible with respect to kinematic constraints

Spring-loaded doors have additional constraints:

1. Once contact is made, some part of the robot is in contact with the door at all times

2. Keeping the door open cannot violate joint torque limits

Our planning algorithm operates by constructing and searching a graph of predefined and dynamically generated motion primitives [23]. The graph search uses the constructed graph to find a path from the start state (corresponding to the current position of the robot with respect to the door and the current door angle) to *any* state satisfying the goal conditions, specifically opening the door such that the robot can pass through the door frame by moving forward.

In the following sections, we explain the algorithm, covering the state-space representation, motion primitives, cost function and heuristics, and graph search.

5.2.1 Graph Representation

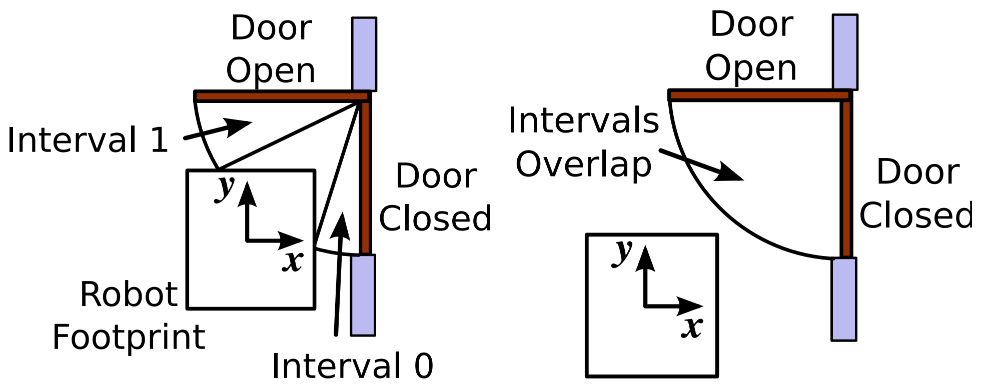
The graph is constructed using a lattice-based representation, as described in Chapter 3. A lattice is a discretization of the configuration space into a set of states and connections between those states, where every connection represents a feasible path.

With respect to the framework of Chapter 4, and without loss of generality, let us consider the reduced-dimensional graph without the the superscript l for cleaner notation. Let $G = (S, T)$ denote the graph G we construct, with S the set of states and T the set of transitions between states. To discuss the states in S, let us first consider the motion of a mobile manipulator opening a door. Let $(x_b, y_b, \theta_b) \subset SE(2)$ represent the configuration of the base, and $\theta_d \subset \mathbb{R}$ the set of possible door angles. One additional variable is needed to store the free angle of the manipulator. This produces states of 5 continuous variables. Storing the side of the door the robot is contacting, as well as the part of the robot in contact with the door, takes additional, though discrete, variables. We consider right end-effector on handle, left end-effector on handle, and base against door contacts.

As we mentioned in [21], it is sufficient to use a more compact representation of the door angle. Instead of storing the door angle θ_d directly, we utilize a discrete variable, d, called the *door interval*. Door intervals are illustrated in Figure 5.2. The door interval is 0 when the door is at an angle where it may be fully closed without colliding with the robot. A door interval of 1 denotes that the door is at an angle where it may be fully opened without colliding with the base. The two intervals are separate if the body of the robot intersects the swept area of the door, as is the case in Figure 5.2a. If the robot is far enough from the door, as in Figure 5.2b, these intervals overlap, denoted with a value of 2. We note that, though the door angle is not stored, the planner has the ability to quickly reconstruct the set of door angles for a robot pose p, $\Lambda(p)$, feasible given the current contacts.

Additionally, instead of storing the free angle of the manipulator, it is sufficient to place restrictions on the manipulator. We chose to restrict it to elbow-down configurations. Conservative collision checking can be done against the swept volume of the arm, which can be precomputed for end-effector poses. In our compact representation, a state in the state-space used by the planner, $s \in S$, is given by

$$s = (x_b, y_b, \theta_b, d, h, v)$$



(a) (b)

Figure 5.2: Door intervals do not overlap when the robot intersects the area swept by the door (a), but do overlap as the robot moves further away (b).

where d is the door interval, h is the side of the door whose handle is being grasped, and v indicates the part of the robot in contact with the door. v takes 4 possible values, corresponding to no contact, left end-effector on door handle, right end-effector on door handle, and base against door.

With respect to the framework of Chapter 4, any state in the full-dimensional state-space, $s^f \in S^f$, can thus be represented by including the corresponding arm angles, and is given by

$$s^f = (\underbrace{x_b, y_b, \theta_b}_{\mathcal{X}}, \underbrace{\theta_1, \ldots, \theta_n}_{\mathcal{Y}}, \underbrace{d, h}_{\mathcal{Z}}, \underbrace{v}_{\mathcal{W}})$$

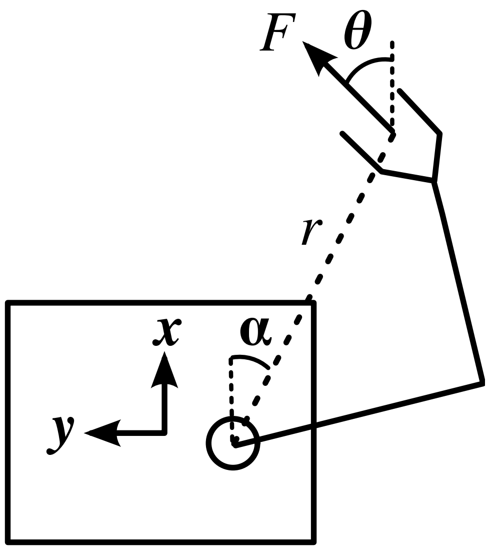
We use a lattice-based planning representation [65,88] to define the set of transitions T between states. A motion primitive is a discretized, finite-length feasible path between neighboring states. It can be defined as a discretized path of intermediate offsets of (x_b, y_b, θ_b) and transitions in d, h, v, or some subset thereof. The lattice graph is dynamically constructed by the graph search as it expands states.

We use two different types of motion primitives that connect a state s to a successor state, $s' \in succ(s)$. The first primitives describe motions for the mobile base. For a holonomic base they represent forward and backward translational motion, rotation in place, strafing, and moving forward and backward while turning. For a nonholonomic base, they satisfy the nonholonomic constraints on its motion. These primitives are augmented with transitions between door interval values d, generated at runtime because changes in d are a result of moving the base with respect to the door. The second set of primitives do not include motion for the base. Instead, these are transitions between discrete variables representing the part of the robot in contact with the door v and the side of the door being grasped h. Because these primitives do not include motion of the mobile base, they cannot transition between values for the door interval d.

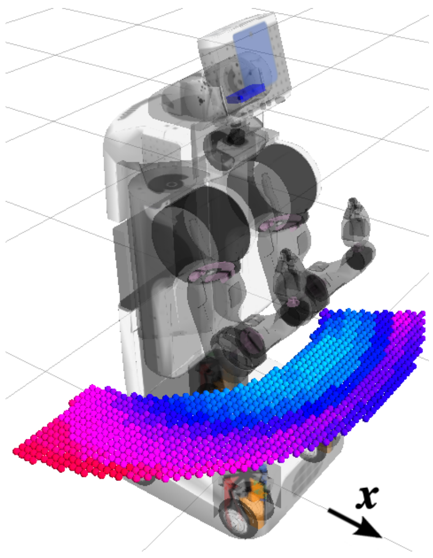
Before a successor of state s, s', can be added to the lattice graph, it must first be checked for feasibility. For a successor to be valid, for every pose p along the discretized motion primitive, the set of valid door angles $\Lambda(p)$ must overlap between adjacent poses. The corresponding door intervals for adjacent poses must also be the same (or include the overlapping door interval, $d = 2$). This way, the base can move along the motion primitives from one pose to another while continuously contacting the door. Additionally, the robot base and conservative arm estimate must be collision free, noting that the conservative arm collision check and check of the door handle against the reachable workspace are done when generating $\Lambda(p)$. The allowed contact transitions for the planner are as follows: (1) when the robot is not yet in contact with the door, it may contact the door with either arm or the base, provided the valid door angles for the new contact overlap with the valid door angles of the old contact; (2) when an end-effector is on a handle, the planner may transition to using the other arm on the other side of the door, or bring the base into contact with the door, again provided the valid angles for old and new contact overlap. If the robot is not yet in contact with the door, its only valid angle is the door closed angle. Additionally, if the door is to come into contact with the base, the valid door angles are such that the door is no more than 5 cm from the base.

5.2.2 Precomputation

Finding the valid door angle ranges $\Lambda(s)$ for a given state s can be expensive, motivating moving as much computation as possible offline. To check whether a given door angle is valid requires checking for valid inverse kinematics solutions or checking that the end-effector pose is within the (precomputed) robot's reachable workspace. Further, for spring-loaded doors, the ability to exert a given force normal to the door can be checked by referencing the robot's force workspace, similar to force workspace approach presented in [69]. The force workspace precomputation also functions as a reachable workspace precomputation, returning the reachable workspace when the query is set to regions where the allowable force is greater than zero. For our purposes,



(a)



(b)

Figure 5.3: (a) Values used for the precomputation discretization. α and θ are the angles of the shoulder-handle vector and exerted force vector relative to the x-axis of the base, while r is the magnitude of the shoulder-handle vector. (b) Force workspace computation for the right arm, (for $\alpha \in [-1.9, 0.6]$ and $r \in [0.45, 0.85]$) with color representing the minimum normal force the arm can apply in a given direction, with light blue near the center containing the greatest force at 34.4 N and red at the sides being the least. Values shown for $\theta = 0$.

the force workspace precomputation only has to be done for a horizontal plane at the height of the door handle. It is worth noting that we assume that the door is moving relatively slowly so the quasi-static assumption is appropriate.

We precompute the force workspace values for a range of robot positions and door angles. Possible door handle locations in the robot base frame can be described by three values as shown in Figure 5.3a. The angles α and θ represent the angles of the shoulder-gripper vector and the gripper applied force relative to the x-axis of the base. The distance between shoulder and end-effector is given by r. For a given inverse kinematics solution for the arm, (specifying a value for the free angle) we can calculate the maximum force the arm is able to exert normal to the door in the θ direction. To do so, we require the end-effector Jacobian, defined as $d\mathbf{f}(\mathbf{q})/d\mathbf{q}$ where $\mathbf{f}(\mathbf{q})$ is the forward kinematics solution for the arm given joint angles $\mathbf{q}$.

The end-effector Jacobian is used in the following relation:

$$\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F}_e \tag{5.2.1}$$

where $\boldsymbol{\tau}$ is the vector of joint torques and $\mathbf{F}_e$ is the wrench applied at the end effector. Referring to Figure 5.3, $\mathbf{F}$ is specified as the force component of the wrench $\mathbf{F}_e$ in the direction indicated by the angle θ in a plane parallel to the ground plane.

We exploit the linear relationship between $\boldsymbol{\tau}$ and $\mathbf{F}$ to scale the value of $\mathbf{F}$ by $\min_{i\in joints} \tau_{i,MAX}/\tau_i$, where $\tau_{i,MAX}$ is the upper limit on the torque for joint i, to get the maximum allowed force for that configuration. Because the arm is redundant and multiple inverse kinematics solutions exist for a given desired end-effector pose, this process is repeated for twenty elbow-down inverse kinematics solutions and the minimum force across all these solutions is selected. If no inverse kinematics solution exists, we record the allowed force as zero. Values for $\theta = 0$ are shown in Figure 5.3b.

5.2.3 Cost Function and Heuristic

The cost of a transition in our graph representation is defined as the sum of the two terms,

$$c(s, s') = c_{movement} \times c_{costmap} + c_{door}$$

In the first term, $c_{movement}$ is the cost associated with moving the robot through a given base motion primitive. This term depends on the time it takes to execute the motion primitive. It also allows the user to minimize the use of certain primitives, such as moving backward. $c_{costmap}$ is given by using the maximum cost of the 2-D costmap cells traced by the robot footprint along the transition. The costmap projects world obstacles to a 2-D grid and represents proximity to obstacles with increased costs. The second term, c_{door}, is proportional to the change in door angle nearest the center of the robot's reachable workspace from state to state. To enable this, for each state we record a door angle deemed $\lambda(s)$ which minimizes a function designed to punish deviation from a nominal shoulder-handle distance r_c and angle α_c:

$$\lambda(s) = \min_{\theta_d \in \Lambda(s)} \left(1 - \frac{1}{1 + (r - r_c)^2(\alpha - \alpha_c)^2}\right) \tag{5.2.2}$$

where r and α are defined in Figure 5.3 and r_c and α_c correspond to the center of the reachable workspace for the arm being used. Of course, if the robot has not yet contacted the door or the base is in contact, this term is ignored. Transitions associated with switching between robot-base contacts have fixed costs determined by the user, allowing the user to penalize switching if desired.

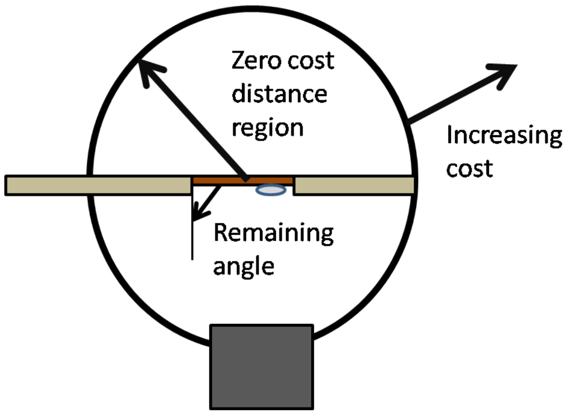


Figure 5.4: Heuristic cost function used in planning. The heuristic penalizes the remaining angle needed to fully open the door and distance to the door. Within a radius of r_{len} of the door, the second term is zero.

The purpose of the heuristic is to guide the search towards a solution using domain-specific knowledge, making it more efficient by reducing the number of unnecessary states explored. The heuristic must be admissible (underestimate the least cost path from the current state to the goal) for the algorithm to return the optimal path; the heuristic must be consistent to ensure that no state is explored more than once. Since part of the condition for a state s to be considered a goal state is that $\Lambda(s)$ overlaps with the fully open door angles, we set the first term of the heuristic to estimate the remaining angle the door needs to be opened before it is considered fully open, $|\lambda(s) - \lambda_{open}|$. The second term in the heuristic is a term for the distance of the robot to a circle of radius r_{len} around the door,

$$\max(0, \sqrt{(x_{robot} - x_{doorcenter})^2 + (y_{robot} - y_{doorcenter})^2} - r_{len}).$$

We set r_{len} to the length of the door from hinge to handle plus the length of the end-effector. The purpose of this term is to estimate the cost of the plan to drive to the door and make contact. Both of these terms are admissible and consistent. When the robot is further than r_{len} from the door, the first term remains at its maximum value. When the robot is able to contact the door, it is closer than r_{len} and so the second term is zero. Because of this, the sum of the terms is an admissible and consistent heuristic.

5.2.4 Search

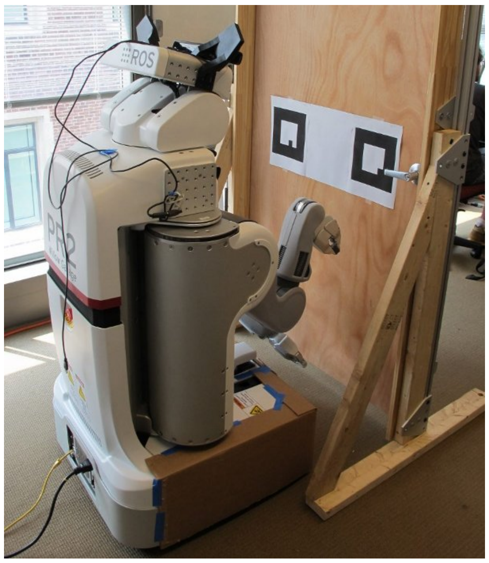
Given a graph as defined above, composed of states linked by motion primitives, we need an efficient way to search it for a solution path. A* search is a very popular method for graph search that finds an optimal path, which may not be possible if deliberation time is limited [46]. Instead, we use an anytime variant, Anytime Repairing A* (ARA*) [66], as described in Chapter 3. The algorithm generates an initial, possibly suboptimal solution then focuses on improving the solution while time remains. The algorithm is provably complete for a given graph G and provides theoretical bounds on suboptimality of solutions. It works by inflating the heuristic by a value $\epsilon \geq 1$. Given additional time, the graph search is able to decrease the bound ϵ to 1.0 and provide the optimal solution.

5.3 Implementation

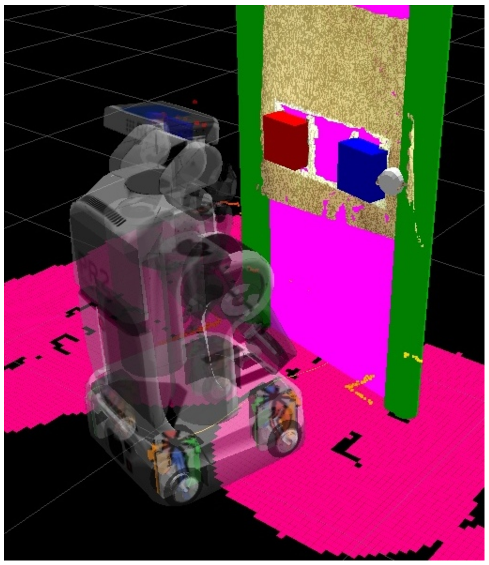
The door-opening task consists of two main stages: first detecting the door and determining its parameters, followed by planning and executing the door-opening motion. Door detection is outside the scope of this work; for determining the door and handle sizes and positions, we use *a priori* knowledge of the door being used either in simulation or real-life trials. The initial position and relative open angle are determined using the ARToolKit [34], which provides a framework for tracking fiducial markers. Each door is measured in advance and several door properties are recorded, including the distance from each marker to the edge of the door, the distance to the door handle, the depth of the door handle, the side with the hinge, the direction of swing, and the necessary force at the handle to open the door. We make the assumption that the required force remains constant throughout the trajectory, though [51] shows that the required force diminishes as the door opens.

Our testbed is a Willow Garage PR2, as shown in Figure 5.5. The robot has two arms with 7 degrees-of-freedom, an omni-directional base, a pan-tilt head, and an adjustable height torso. We use a Hokuyo scanning laser rangefinder attached to the base and a tilting laser scanner to generate a 3-D collision map and 2-D costmap for navigation. The left camera of the wide stereo-camera pair is used to detect the ARToolKit markers.

The planner yields a trajectory of n states of the form $s = (x_b, y_b, \theta_b, d, h, v)$. Before the path can be executed on the robot, the inverse kinematics at each position must be



(a)



(b)

Figure 5.5: ARToolKit detection of the door using one of the wide-stereo cameras (a) and a visualization of the door (b). The inflated 2-D costmap is in pink.

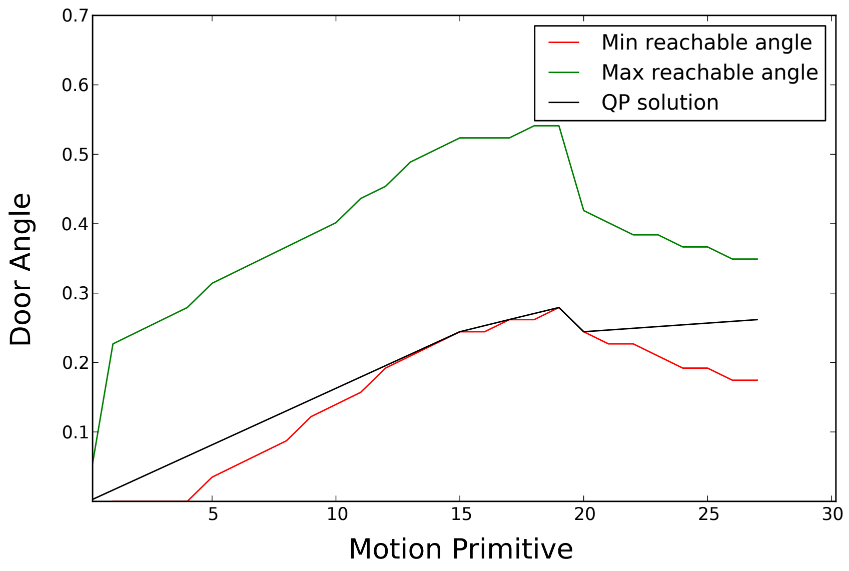


Figure 5.6: The door angle trajectory from the the quadratic problem solution, plotted alongside the maximum and minimum reachable door angles at each step.

resolved, requiring the door angles. Because for each state there may be many feasible door angles in a given door interval, it is desirable to minimize the motion of the door. We formulate and solve the following quadratic problem as a post-processing step:

$$\begin{aligned} &\min \textstyle\sum_{i \leq n} ||\theta_i - \theta_{i-1}||_2^2 \\ &s.t.\ \theta_{i,LB} < \theta_i < \theta_{i,UB} \end{aligned} \tag{5.3.1}$$

solving for the door angle at each step i, where $\theta_{i,UB}$ and $\theta_{i,LB}$ are the upper and lower bounds are given by maximum and minimum angles in $\Lambda(s_i)$. An example solution is shown in Figure 5.6. With the door angles known, the door handle locations and thus end-effector poses are known. Combined with the trajectory for the base, we have sufficient information to generate the joint space trajectory for the arms by solving the inverse kinematics for the arm at each step (or at least those steps in which an arm is attached to the door). For any states in which an arm is not attached to the door, we assume the arm joint angles remain unchanged from the previous state. Because of the force workspace precomputation and conservative collision checking, we know an inverse kinematics solution exists. To resolve the kinematic redundancy of the arm, the inverse kinematics solver uses an initial seed value for one of the joint angles, then analytically solves the remaining 6 degrees-of-freedom. In some situations, such as an interior corner door, the presence of walls may separate inverse kinematics solutions into disjoint sets (elbow-up and elbow-down). We handled this by only using elbow-down configurations. Transitions between arms involve calls to an arm-specific planner using sampling-based motion planning [26]. At this point, a trajectory for the base and joint angle trajectories for the arms are known; we have mapped the solution from the reduced-dimensional state-space to the full-dimensional space.

5.4 Simulation Results

We have tested our planner on a simulated PR2 and on the physical robot. For all of the experiments, the environments are discretized at a resolution of 2.5 cm. The robot base heading is discretized at 22.5 degrees; 13 motion primitives associated with motion of the base are given for each heading. Path lengths of the primitives range from 2.5 cm to 20 cm. Typical plans were between 20 and 110 motion primitives in length.

Two of our simulated environments are shown in Figure 5.7. The first set of simulated tests, with results in Tables 5.1 and 5.2, shows how the number of states expanded and planning time vary between open and tight spaces. The corner hinge and corner edge environments refer to the narrow hallway environment of Figure 5.7b, minus the right or left wall, respectively. As for the office environment, Figure 5.7a, the obstacle along the left wall is placed at different distances to the door. The corner hinge case is relatively open and simple for the planner to solve; the robot may move through the door as soon as the door open angle is large enough for it to pass through. However, a wall placed on the other side of the door as in the corner edge case requires the robot to open the door further before it can transition to holding the other side of the door and pass through. The narrow hallway is quick to solve because walls on both sides greatly limit the possible states. Conversely, when starting further from the door, as in the office door case shown in Figure 5.8, the fixed cost of transitions to contact the door lead to more states being expanded prior to the contact. With more room (as the obstacle is further and further away from the door) more and more states are expanded. The resulting paths are of decreasing cost as more direct routes from the start to the door become obstacle-free.

The second set of tests, shown in Table 5.3, illustrates the effects of changing the force required to open spring-loaded doors. From the same start state, the plans for 1 N and 10 N are identical; joint torque limits were not a limiting factor at under 10 N. However, moving to 15 N, some of the transitions in the previous plans are infeasible

Door	Planning Time (s)		Expansions		Final Soln. Cost
	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	$\epsilon = 1.0$
Corner Hinge	0.97	2.23	2249	6838	551
Corner Edge	3.32	6.17	11267	25790	880
Narrow Hall	0.13	0.21	873	1160	822
Office (1.0m)	0.74	1.79	2148	6115	707
Office (1.2m)	0.80	2.13	2513	7512	707

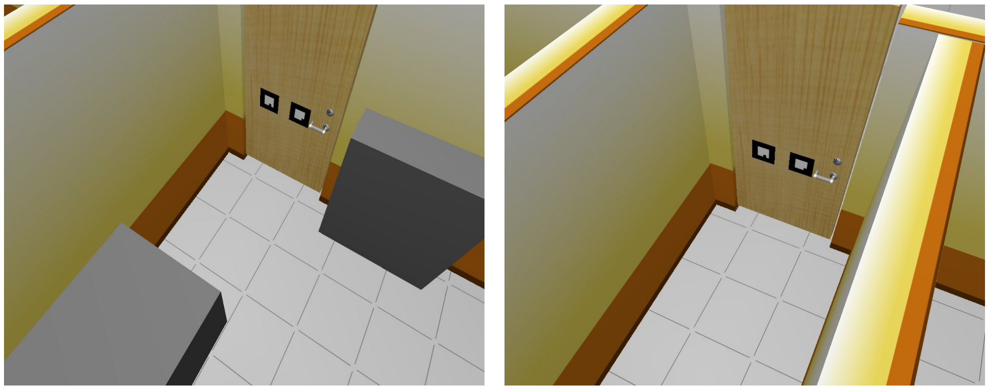
Table 5.1: Simulation: Planning times, expanded states, and costs for **pulling** doors open.

Door	Planning Time (s)		Expansions		Final Soln. Cost
	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	$\epsilon = 1.0$
Corner Hinge	1.07	4.97	7468	31448	775
Corner Edge	4.06	8.79	26516	55134	1076
Narrow Hall	0.17	0.41	1347	1855	750
Office (0.8m)	3.93	6.69	21859	37937	2085
Office (1.0m)	5.03	9.69	26540	48897	1685
Office (1.2m)	6.03	11.11	32879	60032	1463

Table 5.2: Simulation: Planning times, expanded states, and costs for **pushing** doors open.

Normal Force (N)	Planning Time (s)		Expansions		Final Soln. Cost
	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	$\epsilon = 1.0$
1.0	0.97	2.23	2249	6838	551
10.0	1.00	2.24	2249	6838	551
15.0	4.30	5.95	14948	24905	871

Table 5.3: Simulation: Planning times, expanded states, and costs for **pulling** requiring different normal forces at the handle.



(a) (b)

Figure 5.7: Simulated environments include an office door with nearby obstacles (a) and a narrow hallway (b).

so the search must expand more states to route around the infeasible regions. This leads to a longer, higher cost solution. For 20 N and above, no solution exists for pulling open doors; the arms of the PR2 are not strong enough to hold such doors open to allow transitioning from contacts on one side of the door to the other.

5.5 Experimental Results

5.5.1 Experiments at Penn

Next, we discuss results on the physical PR2 for pushing and pulling both spring-loaded and non-spring-loaded doors at the University of Pennsylvania. These results were gathered prior to a recent normalizing of the cost and heuristic functions and have much higher costs. They also specify the initial contact with the door. The results of running twenty planning trials on a few doors for both pulling and pushing are

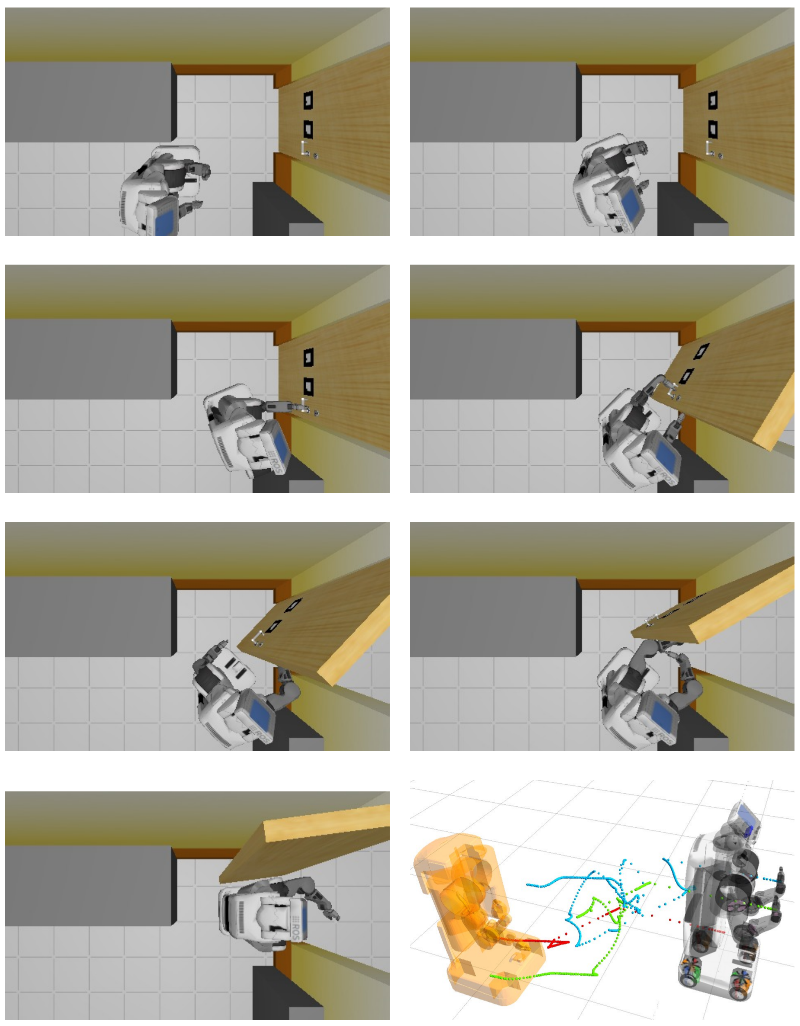


Figure 5.8: Frames from a successful test in the office environment. The left obstacle is 1.2 m from the door frame. The last image shows a trace of the base and end-effector locations throughout the motion. Note from the base trajectory (red) that the robot backs up and turns as it initially pulls open the door.

listed in Table 5.4 and Table 5.5, respectively. The testbed (shown in Figure 5.5a) and conference room door were not spring-loaded. The kitchen door required 15 N normal force at the handle to open, while the office door required 27 N. In between trials, we moved the starting location of the base by a few centimeters and the initial orientation varied but was kept within ±20 degrees of normal to the door. On average, both pushing and pulling plans took under 6 seconds to find an initial solution, in most cases, much less. The cost of pushing plans is higher, as the robot must plan through a narrow passageway in the costmap, most likely with nonzero costs. Pulling plans allow the robot to withdraw from the door into open space of the costmap. The testbed door was also narrower than the kitchen door, requiring the robot to pass through higher cost cells in the costmap, but reducing the number of expansions (states added to the graph during planning) and thus yielding faster planning times. The conference room door, the longest to plan, bordered directly on a wall.

Images from the PR2 pushing open a spring-loaded door and pulling a non-spring-loaded door are shown in Figures 5.9 and 5.10. Twenty trials total were carried out past the planning stage. The gripper managed to slide off the handle when pushing the kitchen door open, but the robot was able to successfully pass through the door; these trials have been counted as successes. As long as the robot was initially able to grasp the door handle (i.e., aside from door detection issues), the robot never failed to pull open a door which required 15 N or less normal force at the handle, and only once failed to push a door requiring less than 27 N of normal force at the handle. The

Door (Force, N)	Planning Time (s) First Soln. $\epsilon = 5.0$	Planning Time (s) Final Soln. $\epsilon = 1.0$	Expansions First Soln. $\epsilon = 5.0$	Expansions Final Soln. $\epsilon = 1.0$	Final Soln. Cost $\epsilon = 1.0$
Testbed (0)	0.249 ± 0.201	0.485 ± 0.244	79.8 ± 75.0	99.3 ± 68.6	135,340 $\pm 141,760$
Kitchen (15)	2.22 ± 1.59	2.95 ± 2.10	370 ± 265	465 ± 347	12,841 $\pm 7,683$
Conference (0)	3.50 ± 1.06	5.59 ± 2.76	601 ± 175	942 ± 486	29,032 $\pm 21,696$

Table 5.4: Planning times, expanded states, and costs for **pulling** doors open. Contains averages and standard deviations for 20 trials on each door.

Door (Force, N)	Planning Time (s) First Soln. $\epsilon = 5.0$	Planning Time (s) Final Soln. $\epsilon = 1.0$	Expansions First Soln. $\epsilon = 5.0$	Expansions Final Soln. $\epsilon = 1.0$	Final Soln. Cost $\epsilon = 1.0$
Testbed (0)	0.292 ± 0.069	2.04 ± 0.467	86.2 ± 20.8	649 ± 162	418,310 $\pm 38,174$
Kitchen (15)	2.20 ± 0.301	3.03 ± 0.245	916 ± 116	1,228 ± 80.9	94,809 $\pm 70,150$
Office (27)	0.636 ± 0.132	1.92 ± 0.251	195 ± 41.6	589.6 ± 91.8	441,890 $\pm 61,484$

Table 5.5: Planning times, expanded states, and costs for **pushing** doors open. Contains averages and standard deviations for 20 trials on each door.

one failure, due to an issue with the costmap, generated a path which collided with the door frame.

Failures outside the scope of the planning occurred due to poorly estimated door parameters. Such errors resulted in missed grasps of the door handle, as happened in eight additional trials. Errors in handle and hinge detection can generate large internal forces in the arm during the motion; the end-effector slipped off the door handle in three successful trials.

5.5.2 Experiments at Carnegie Mellon University

The following trials were carried out at Carnegie Mellon University. The trials at multiple universities illustrate the robustness and validation of the door opening solution we have presented. Trials were carried out on 5 unique doors inside room 1612 Newell-Simon Hall at Carnegie Mellon's campus. Due to space limitations around each door, not every door was tested for both pushing and pulling. Two non-spring-loaded doors were pulled open, three non-spring-loaded doors were pushed open, and one spring-loaded door was also pushed open. The spring-loaded door was not pulled open because the PR2 arms were too weak to do so. Detailed results from the planner (i.e., solution cost and number of state expansions during the search) were not recorded; total execution time was recorded.

The PR2 was able to successfully open each type of door. 22 of 31 trials were successful in opening the door and moving the robot through. Four of the failures occurred during the 15 trials of pushing a non-spring-loaded door (11 of 15 = 73% success rate). One failure occurred during the 5 trials on the spring-loaded door. Four of the failures were on the 11 trials on pulling non-spring-loaded doors (7 of 11 = 64% success rate). Of the above failures, 2 were caused by the robot colliding with the door frame (in one case, the base collided, while in the other it was the arm). The rest of the failures were caused by incorrect door localization, not our area of research.

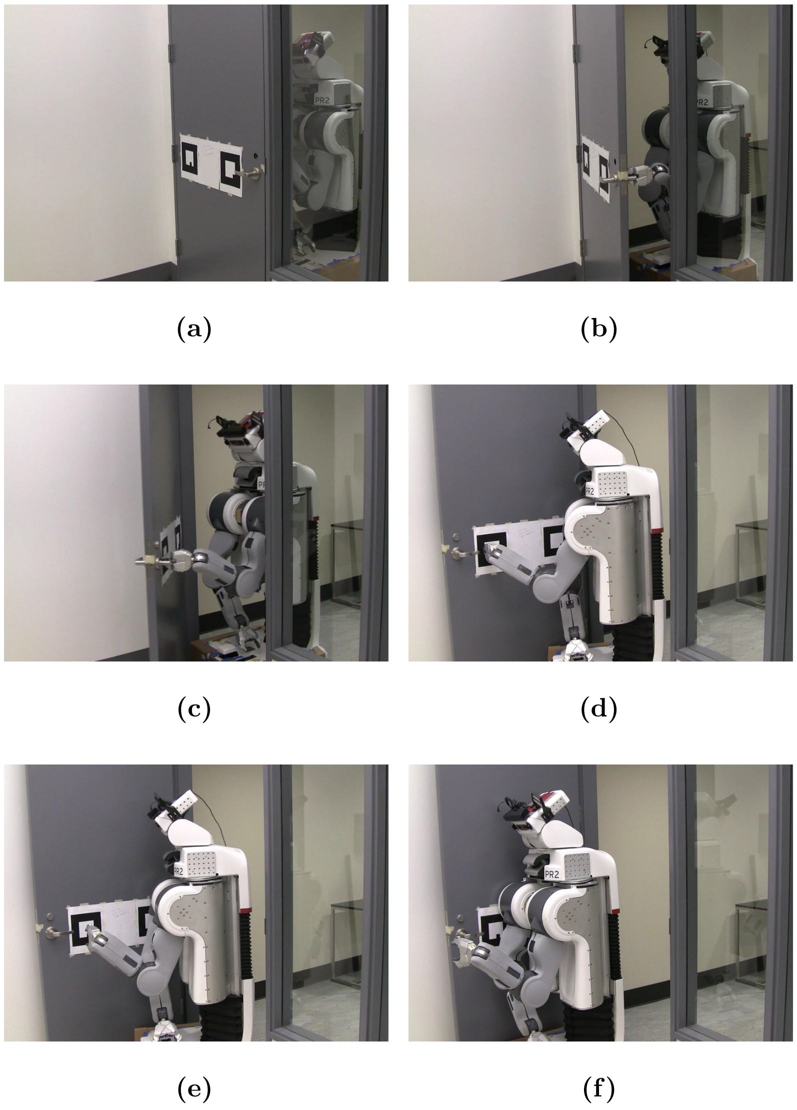
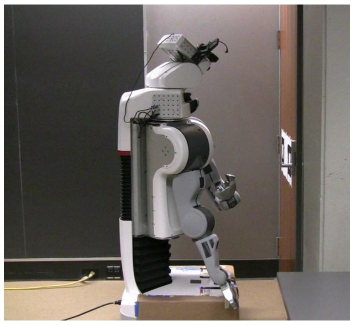
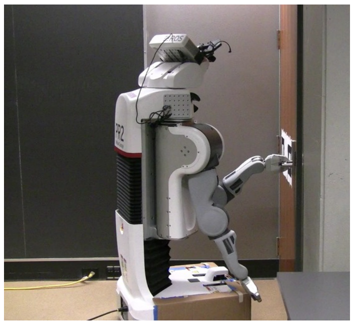
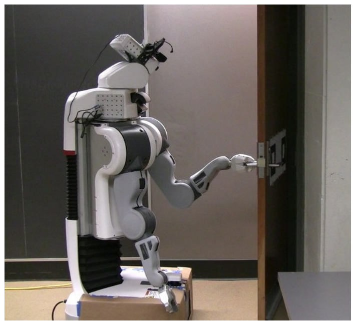
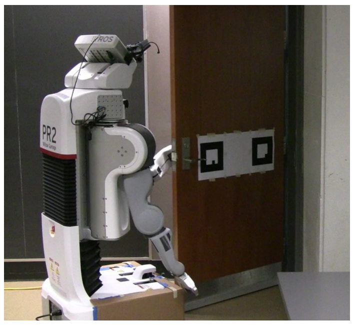


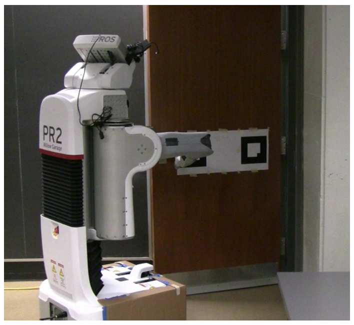
Figure 5.9: Image sequence from pushing open a kitchen door requiring 15 N normal force at the handle (a-f). Initially the left arm is on the door handle (a-d), then the plan transitions to pushing the door with the base (e-f).

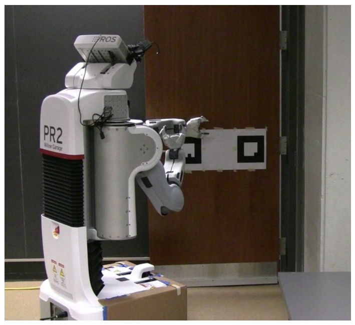

(a)

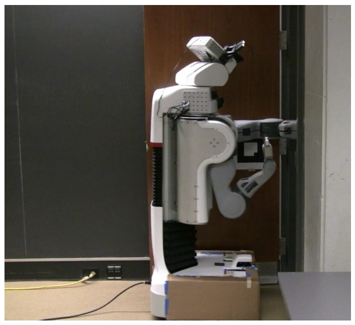

(b)


(c)


(d)


(e)


(f)


(g)

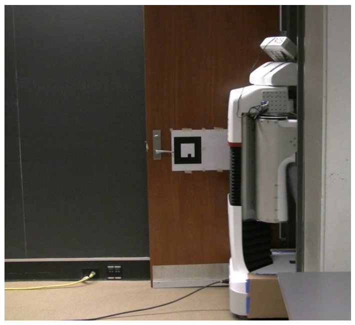

(h)

Figure 5.10: Image sequence from pulling open a non-spring-loaded door (a-h). The robot is initially not contacting the door (a), makes contact with the left arm and begins pulling the door open (b-d), transitions to contact using the right arm (e), then transitions to bracing with the left arm (f-g) and then base as it moves through the door (h).

The time from detecting the door to grasping the handle ranged from 24 to 51 seconds, for an across-the-board average of 39 seconds. Planning times were typically under 3 seconds, taking up to a maximum of 6. When pushing, the average plan time was 3.1 seconds; the average time for pulling was 1.3 seconds. Spring-loaded pushing was an average of 4.3 seconds. The length of the planning time is heavily influenced by the obstructions near the door; a narrower region to plan in (as long as it contains a feasible path) means fewer possibilities to expand and often a lower planning time. Total execution times ranged from 84 to 145 seconds. Pulling trials, with the scripted arm transition at the end, took longer than pushing, averaging 131 seconds compared to 94. The spring-loaded pushed door averaged 85 seconds from start to finish.

5.6 Discussion

Detection of the ARToolkit markers turned out to be a nontrivial problem. Image thresholding parameters had to be adjusted based on light levels in different rooms. Even so, incorrect or slightly off detections resulted in the failure to grasp the door handle, the cause of 7 of the 9 failures at CMU and 8 unsuccessful grasps at Penn. Additionally, the controller framework of the PR2 caused difficulties synchronizing the execution of arm and base motions. A half-second delay for the base motion was empirically chosen to address this. The trajectories generated for the PR2 include zero joint velocity for the arms after each motion. Even with smoothing, the motions are still noticeably jerky, but current inability to smooth the motion of the base and arms in a linked fashion limits the amount of smoothing possible. Because the planning is done for the door and robot base, with the arm configurations generated afterward using inverse kinematics, there are sometimes issues with the arm contacting the door frame. An approximation to the arm configuration is used when planning, which mitigates this issue, but integration of the inverse kinematics into the planning will address the issue.

We showed that the PR2 could be used to pull open spring-loaded doors up to roughly 20 N, but spring-loaded doors often require around 60 N to open [51]. When using a PR2 or similar robot, alternative approaches must be employed. For instance, the robot could pull open much heavier doors by extending the arm to a singular configuration so the force at the end-effector then depends on the strength of the base. However, the robot would then have to release the door and quickly move through to make sure it passes through the door before it closes. This could be used in concert with dynamic motions, whipping the door open and immediately moving through. Another similar approach wraps the arm around the robot's front, so the robot is facing away from the door with its end-effector on the handle, bracing the arm so that the base is used to pull open the door [1]. The arm can be unwrapped and the body used to brace the door as the robot passes through.

This work marks the first time that door opening with changing chain topologies has been handled by a single planner, including which transitions to include and when to include them. We have demonstrated the effectiveness of our framework; this represents a significant step in our ability to generate multi-contact and switching-contact plans, all while maintaining the repeatability and optimality of search-based planning that make it attractive for use in human environments.

Chapter 6

Application to Walking

This chapter addresses control and planning for humanoid balancing and quasistatic locomotion. As the robot walks, closed chains are created when both feet are on the ground and destroyed as soon as one foot is no longer in contact. The motion of the robot can be described by the locations of the feet, abstracting away the complexity of the closed chain of the lower body, making it a good fit for our framework. On that note, our framework inspires our work on walking, but the system is complex and the required assumptions and thus the guarantees of the framework do not apply directly.

This work was motivated by our participation in the DARPA Robotics Challenge, a research competition with the goal of advancing humanoid robotics [29]. We used an Atlas robot developed by Boston Dynamics [12], an approximately human-sized humanoid, representative of a growing number of humanoid robot platforms. The first round of the competition used a simulated Atlas robot; this is where we focused our efforts. The Virtual Robotics Competition (VRC) featured three tasks; entering and driving a vehicle, locomoting across rough terrain (including a mud pit, hills, and strewn cinderblocks) and manipulation (attaching a hose to a standpipe and turning a valve). The balancing controller herein is used for our manipulation and the initial phases of our vehicle ingress. The walking controller is used for crossing rough terrain.

We present modifications to a grasping-inspired balancing controller developed in [78] which allow it to work on the Atlas humanoid. We then present an extension to quasistatic walking, able to follow a given trajectory for the center of mass or given footstep locations. The primary benefit of the quasistatic approach over a dynamic walking gait is that the approach is robust to walking on unknown terrain as well as poor force sensing, albeit at the cost of overall movement speed.

6.1 Atlas Humanoid

The simulated Atlas humanoid developed by Boston Dynamics is shown in Figure 6.1 [12]. It features 6 degrees-of-freedom in each arm, 3 at the shoulder, 1 at the elbow, and 2 in the wrist (roll and pitch). Similarly, each leg has 6 degrees-of-freedom, with 3 at the hip, 1 at the knee, and roll and pitch at the ankle. The total mass of the robot is approximately 98 kg, 63 kg of which corresponds to the upper body. The robot has inertial measurement units (IMUs) mounted in the pelvis and head as well as head-mounted stereo cameras and scanning laser rangefinder. The ankles contain a multi-axis force-torque sensor, measuring the the force along the ankle Z-axis and the

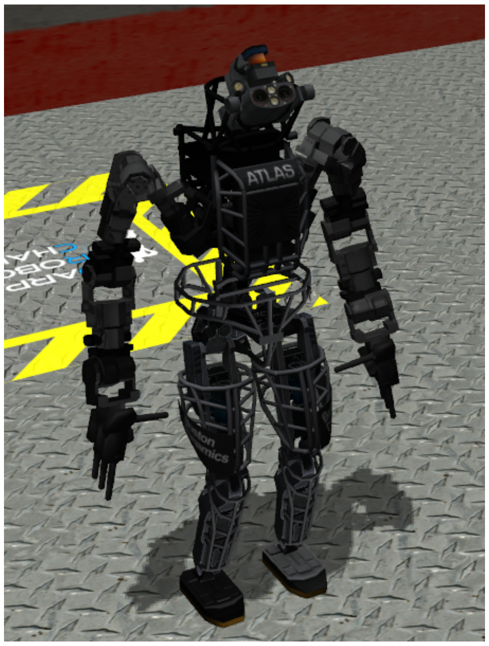


Figure 6.1: The simulated Atlas robot developed by Boston Dynamics.

torque along the roll and pitch. The feet do not contain pressure or contact sensors (at least not in the simulated version of Atlas).

6.2 Background

For a robot to statically balance, the projection of the center-of-mass to a plane perpendicular to the gravity vector (for simplicity, we will call this the ground plane) must lie within the support polygon of the robot. For a robot on flat ground, the support polygon can be defined as the convex hull of all contacts with the ground. When frictional contacts are made upon sloped surfaces, the center of mass must lie above a nonlinear convex set that depends on the properties of the contacts; see [14] for a detailed treatment.

A majority of biped walking and balancing works rely on Zero Moment Point (ZMP), surveyed in [108]. The gist of the ZMP method is that for the point on the foot where the ground reaction force is acting, the the roll and pitch components of the moment must be zero. Force sensors in the feet are typically used in order to implement a ZMP control loop and ensure the desired contact state between the robot and ground is maintained. This allows the robot to dynamically balance and adapt to unknown external perturbations. Balancing controllers based on ZMP are found in [50, 75, 100]. In this work we do not use ZMP methods, but rather a balancing approach adapted from the field of robot grasping; the choice was motivated by the lack of required sensors for ZMP control on the feet of the simulated robot.

6.3 Balancing Controller

We will first outline the balancing controller described in [78]. The method is based on frictional grasping; forces $\mathbf{f}$ are applied at contact points $\mathbf{P}$ to generate a net wrench $\mathbf{F}$ on the on the object being grasped sufficient to keep it restrained. In the case of balancing, the desired wrench is applied to the robot center of mass (COM) and is used to track a desired pelvis orientation and COM location; i.e., the robot should remain relatively upright, compensating for gravity, with its projected COM within the support polygon. The contact forces used to do this are those on the feet of the robot.

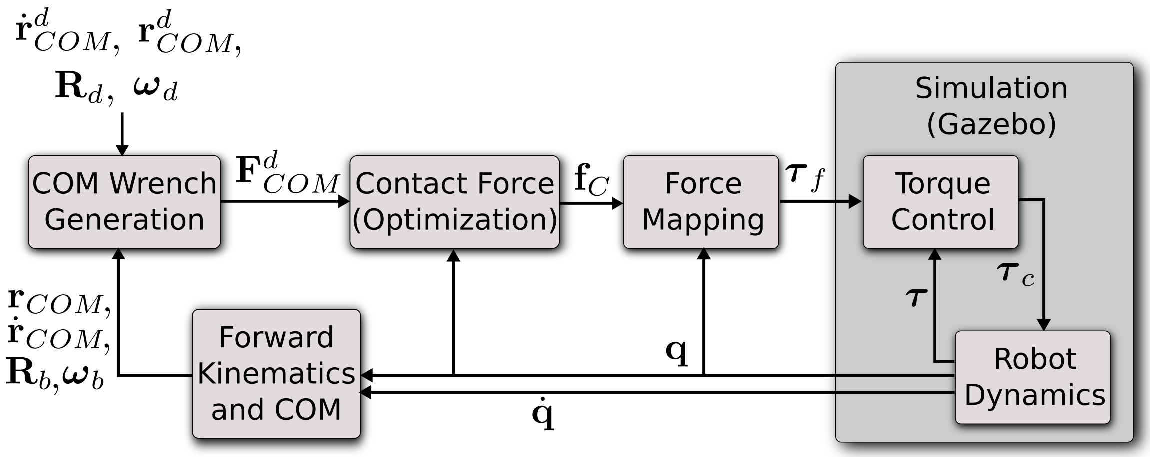


Figure 6.2: Overview of the balancing controller.

6.3.1 Center of Mass Position and Posture Controller

The desired center of mass (COM) force is given by

$$\mathbf{f}^d_{COM} = m\mathbf{g} - \mathbf{K}_P(\mathbf{r}_{COM} - \mathbf{r}^d_{COM}) - \mathbf{K}_D(\dot{\mathbf{r}}_{COM} - \dot{\mathbf{r}}^d_{COM})$$

where the gravity compensation term contains m the total mass of the robot and $\mathbf{g}$ the gravity vector, while the latter terms are a PD feedback law to drive the COM to a desired location. $\mathbf{K}_P$, $\mathbf{K}_D > 0$ are proportional and differential gain matrices, and $\mathbf{r}^d_{COM}$, $\dot{\mathbf{r}}^d_{COM}$ are the desired position and velocity of the COM.

The desired COM torque is used to track a desired pelvis orientation. Let $\mathbf{R}_b$ be the current and $\mathbf{R}_d$ be the desired pelvis orientation. From the quaternion representation of $\mathbf{R}_d^T\mathbf{R}_b = (x, y, z, w)$, let $\delta = w$ and $\epsilon = (x, y, z)$. Then an orientation controller for pelvis orientation is given by

$$\boldsymbol{\tau}^d_{COM} = -\mathbf{R}_b(2(\delta\mathbf{I} + \hat{\epsilon})\mathbf{K}_r\epsilon + \mathbf{D}_r(\omega - \omega^d))$$

where $\mathbf{K}_r$, $\mathbf{D}_r$ are symmetric, positive definite stiffness and damping matrices, respectively. This controller acts as a damped spring to align the current orientation $\mathbf{R}_b$ with the desired $\mathbf{R}_d$, as shown in [16]. Together, $\mathbf{f}^d_{COM}$ and $\boldsymbol{\tau}^d_{COM}$ comprise the desired COM wrench, $\mathbf{F}^d_{COM}$.

6.3.2 Contact Force Distribution

Now that we have a desired wrench to apply at the COM, we need to find the contact forces at the feet that will produce it. The following is a brief review of multi-contact grasping. The contact forces at the feet are subject to the positivity restriction; they can push but not pull the ground. Coulomb's friction model is used, stating that the contacts do not slip when

$$f^t \leq \mu f^n$$

where f^n is the magnitude of the normal component of the contact force, f^t the tangential component, and μ the coefficient of friction. In $\mathbb{R}^3$, this restricts the set of allowable contact forces to a cone called the friction cone, whose axis is along the surface normal with a semi-angle of $\phi = \text{atan}(\mu)$.

The total wrench on the object, $\mathbf{F}_o$, is the sum of the wrenches from all of the contacts expressed in the object's coordinate frame, O. For a system with η contacts, let $\mathbf{f}_c$ be a vector stacking all the individual contact forces, $\mathbf{f}_c = (\mathbf{f}_1 \cdots \mathbf{f}_\eta)^T$. Then the expression for the total wrench is

$$\mathbf{F}_o = \mathbf{G}\mathbf{f}_c$$

where G is the grasp map, mapping the wrenches from the local contact point coordinate frame P_i to the object frame O and multiplying by the wrench basis characterizing the contact model. For more information, see [73]. With all frictional point contacts, the grasp map becomes

$$\mathbf{G} = \begin{pmatrix} \mathbf{R}_{p_1} & \cdots & \mathbf{R}_{p_\eta} \\ \hat{\mathbf{r}}_{p_1}\mathbf{R}_{p_1} & \cdots & \hat{\mathbf{r}}_{p_\eta}\mathbf{R}_{p_\eta} \end{pmatrix}$$

where $\mathbf{R}_{p_i}$ and $\hat{\mathbf{r}}_{p_i}$ represent the orientation and cross product matrix of the position of the contact i in the object reference frame O.

When standing, the grasp map $\mathbf{G}$ is known and we need to solve for the contact forces $\mathbf{f}_c$ at the feet. Because the problem is underconstrained, we cast this problem as a quadratic optimization.

$$\min \alpha_1 ||\mathbf{F}^d_{COM} - \mathbf{G}_{COM}\mathbf{f}_c||_2^2 + \alpha_2 \mathbf{f}_c^T \mathbf{f}_c$$

$$s.t.\ \mathbf{f}_{c_i} = \textstyle\sum_{j=1}^{k} \sigma_{ij}\mathbf{n}_{ij}, \sigma_{ij} \geq 0 \quad i = 1 \ldots \eta$$

The constraints above come from approximating the friction cone as a polyhedron; $\mathbf{n}_{ij}$ is the j-th edge of the convex cone at the i-th contact point. The first term of the cost function penalizes distance between the effective COM wrench $\mathbf{F}_{COM} = \mathbf{G}_{COM}\mathbf{f}_c$ and the desired COM wrench; the second term attempts to evenly distribute the contact forces. Weights α_1 and α_2 are chosen such that $\alpha_1 >> \alpha_2 > 0$.

Now that we have the contact forces at the feet, we can find the equivalent wrenches in each foot's frame. The wrenches can then be mapped to joint torques using the

Jacobian for each leg, using

$$\mathbf{J}_c = \mathbf{J}^b - \mathbf{R}_{FP}\mathbf{J}_{COM}$$

$$\boldsymbol{\tau} = \mathbf{J}_c^T\mathbf{F}$$

where $\mathbf{J}^b$ is the body Jacobian for each foot with the pelvis as the root link, $\mathbf{R}_{FP}$ is the rotation from foot to pelvis, $\mathbf{J}_{COM}$ the center of mass Jacobian for each leg, and $\boldsymbol{\tau}$ the joint torque vector.

6.3.3 Balancing for Manipulation

The balancing controller is designed to keep the COM in the desired location, but because the COM location is a function of all joint angles, the original controller from [78] makes a poor platform upper body manipulation. As the upper body reconfigures, the lower body must adjust and move the pelvis to keep the COM over the support polygon, as shown in Figure 6.3. The solution is to wrap another feedback control loop around the pelvis position. Rather than the COM wrench generation tracking the center of the support polygon and the desired COM height, feedback control is used to drive the COM to a set point based on the pelvis tracking error. Thus, the modified controller attempts to maintain the desired pelvis pose by allowing the projected center of mass to traverse the support polygon. If the projected center of mass is about to leave the support polygon, the pelvis is allowed to move to keep the robot balanced.

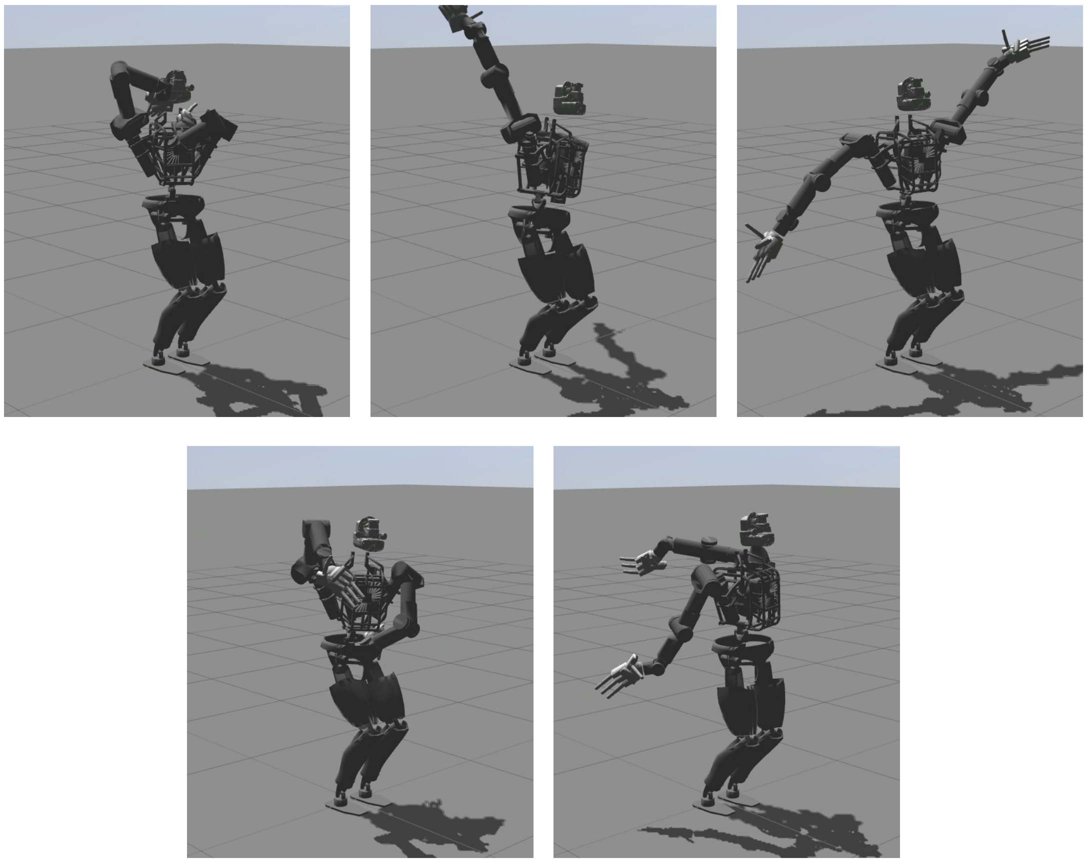


Figure 6.3: Testing the balancing controller by moving the arms to random configurations. The desired orientation of the pelvis is vertical and the desired center of mass location is at the geometric center of the support polygon. The lower body adapts to keep the center of mass from shifting more than 3.4 cm during the run.

6.3.4 Implementation Details

The controller requires measurement of the robot's COM and pelvis orientation with respect to the world frame. The orientation is given by an inertial measurement unit (IMU) in the pelvis. Forward kinematics are used to determine the location of the robot COM with respect to the stance foot location, and hence the support polygon.

The following gain matrix values are used the balancing controller:

$$K_r = \begin{pmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{pmatrix} \qquad D_r = \begin{pmatrix} 150 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 150 \end{pmatrix}$$

$$K_p = \begin{pmatrix} 3000 & 0 & 0 \\ 0 & 4000 & 0 \\ 0 & 0 & 1500 \end{pmatrix} \qquad K_d = \begin{pmatrix} 1500 & 0 & 0 \\ 0 & 1500 & 0 \\ 0 & 0 & 1000 \end{pmatrix}$$

The constrained optimization problem to determine forces at contact points was solved using CVXGEN, an online tool that generates fast, custom C code for small, QP-representable convex optimization problems [71]. We used weights $\alpha_1 = 0.99$ and $\alpha_2 = 0.01$ for the optimization. The friction cone approximation used is given as:

$$n_1 = \begin{pmatrix} -0.0990 \\ -0.0990 \\ 0.9901 \end{pmatrix} \quad n_2 = \begin{pmatrix} 0.0990 \\ -0.0990 \\ 0.9901 \end{pmatrix} \quad n_3 = \begin{pmatrix} 0.0990 \\ 0.0990 \\ 0.9901 \end{pmatrix} \quad n_4 = \begin{pmatrix} -0.0990 \\ 0.0990 \\ 0.9901 \end{pmatrix}$$

6.4 Extension to Walking

The simulated Atlas contains no contact or pressure sensors on the foot; the only sensors present are in the ankle and they provide roll and pitch torque as well as force along the ankle positive Z-axis. The relative lack of sensory information from the foot makes calculating the center of pressure, needed for implementing ZMP walking, difficult. Additionally, quasistatic walking is well suited to imperfect knowledge of the terrain ahead of the robot. Because the COM remains over the stance foot until the swing foot is placed, the swing foot is able to conform to the terrain as it is lowered.

6.4.1 Walking State Machine

Walking lends itself well to the use of a state machine, always transitioning from a single-support phase (resting upon the stance foot), to double-support when both feet are on the ground, and back to single-support. As laid out in Figure 6.4, a new footstep command is received when the robot is in double-support. The COM shifts to rest over the new stance foot and the swing foot lifts up and breaks contact with the ground. The swing leg follows a joint trajectory to place the foot into the desired location. As soon as the swing foot makes detectable contact with the ground, we send a query to get the next commanded footstep and enter the double support phase again. If no further command is received, the robot remains in double support. Footsteps were generated at runtime to move the foot forward while rotating it to track a desired heading.

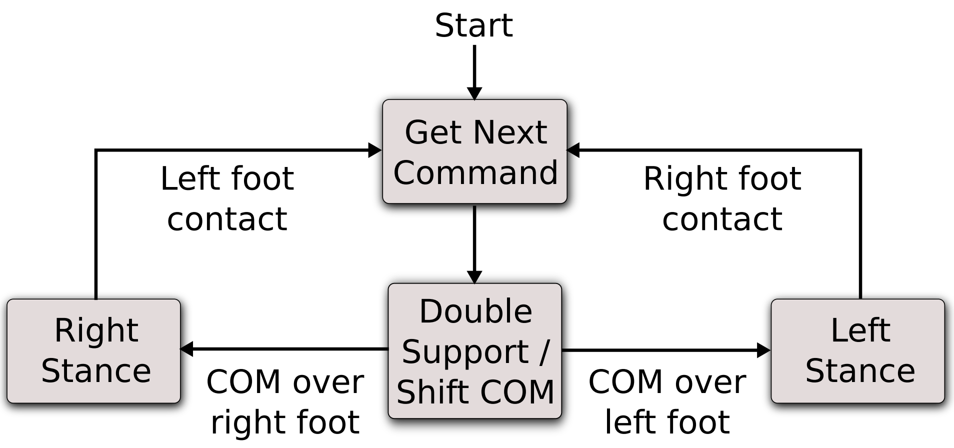


Figure 6.4: Overview of the walking controller state machine.

6.4.2 Implementation Details

The balancing controller utilizes 8-contacts when in double-support and 4-contacts during single support. Contacts are located at the corners of the feet. In order to minimize the difference in contact forces when adding or removing contacts, we modify the objective function to add in a term that penalizes the difference between the contact forces from the previous controller iteration:

$$\min\ \alpha_1||\mathbf{F}^d_{COM} - \mathbf{G}_{COM}\mathbf{f}_c||^2_2 + \alpha_2\mathbf{f}_c^T\mathbf{f}_c + \alpha_3||\mathbf{f}_{c_{prev}} - \mathbf{f}_c||^2_2$$

$$s.t.\ \mathbf{f}_{c_i} = \textstyle\sum_{j=1}^{k}\sigma_{ij}\mathbf{n}_{ij}, \sigma_{ij} \geq 0 \quad i = 1\ldots\eta$$

In practice, we used weights $\alpha_1 = 0.9$, $\alpha_2 = 0.01$, and $\alpha_3 = 0.09$. To handle the foot transitions, the previous contact forces for a given foot were defined as zero if the foot was not in contact with the ground during the previous cycle, and also zero if the foot had just lost contact with the ground (as it could support no weight). In all other cases, the recorded contact forces from the last iteration's optimization were used.

Motion of the free leg can be achieved by planning a desired trajectory for the full leg, using inverse kinematics with a desired foot trajectory, or using Jacobian motions. We chose to use the Moore-Penrose pseudo-inverse of the Jacobian to update the desired joint angles for the first four joints of the leg each controller iteration.

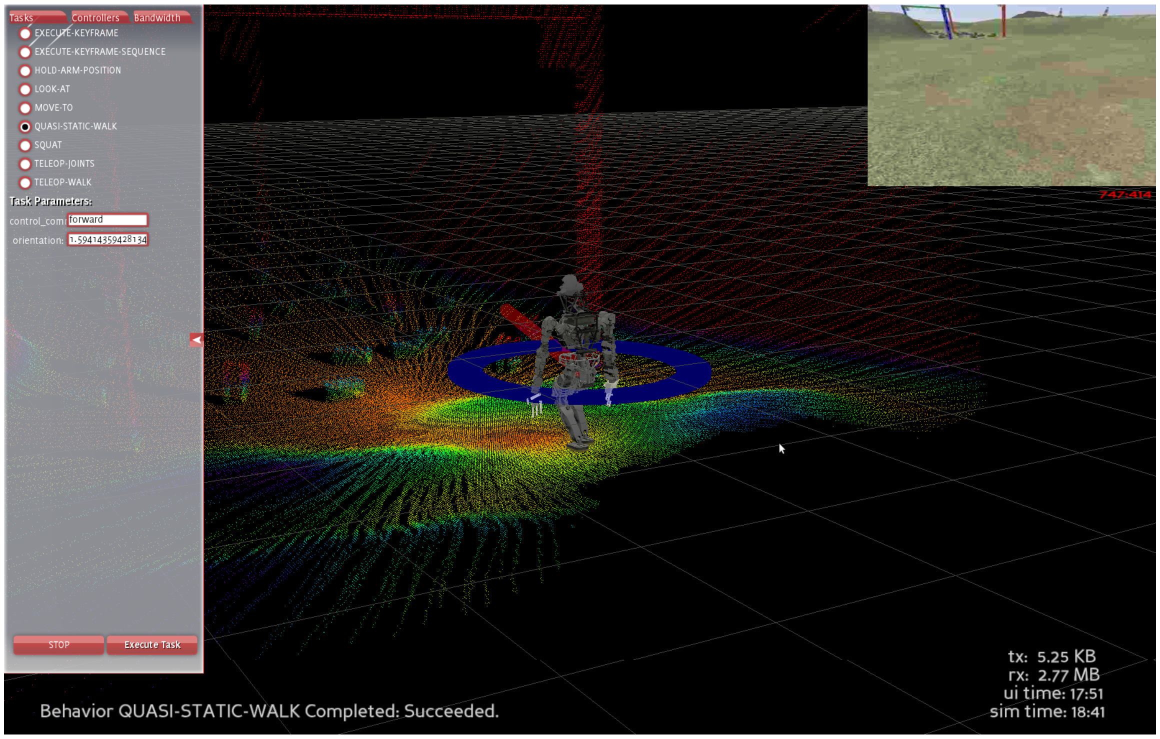
$$d\boldsymbol{\theta} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T d\mathbf{x}$$

The resulting desired joint angles were tracked by a PID plus damping controller. The remaining two joints of the leg correspond to the ankle pitch and roll; these are used to keep the swing foot parallel to the stance foot. Detection of ground contact was achieved by running the ankle force through a low-pass filter with outlier rejection then applying a threshold.

6.5 Results

In our implementation, visual odometry updating at 30Hz was used to localize the robot with respect to the world. The desired yaw for the swing foot was set to the desired heading; the pelvis yaw tracked the midpoint between swing and stance foot yaws. This approach was used in a teleoperation setup for the Virtual Robotics Challenge. The user was able to view a point cloud of the environment and command appropriate heading changes to the robot, as shown in Figure 6.5.

Tracking the heading also allows the Atlas to track desired pelvis trajectories. We do not offer specific guarantees on the tracking of a reference pelvis trajectory. Performance depends on how closely the footsteps follow the turns in the path; i.e., a



[✓ QUASI-STATIC-WALK]

Task Parameters:

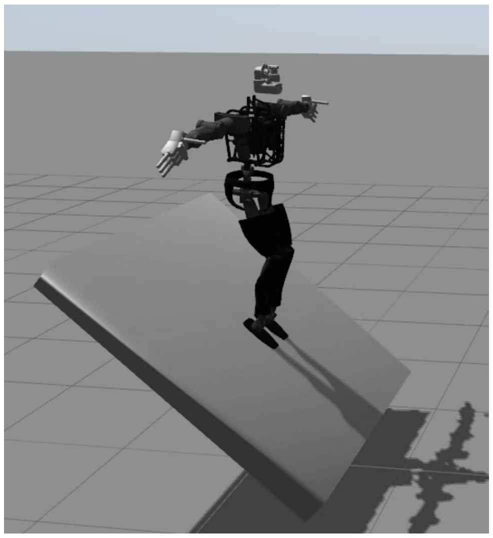
control_comm: forward

orientation: 1.59414359428134

Behavior QUASI-STATIC-WALK Completed: Succeeded.

tx: 5.25 KB
rx: 2.77 MB
ui time: 17:51
sim time: 18:41

Figure 6.5: The user interface shown above allows the operator to specify a desired heading to track. In this mode, Atlas will walk forward along that heading until a new command is received.



(a)

(b)

Figure 6.6: The balancing and walking controllers tested on terrain other than horizontal. In (a) the balancing controller is tested on a platform whose tilt increases over time, failing on a 43 degree slope. In (b) the walking controller is tested walking over hills. The controller expects four contacts with the terrain on each foot; situations with fewer contacts than expected produce jitter.

turn that occurs mid-step will not be tracked as closely as a turn that occurs right at a step. Smaller steps will thus improve tracking performance at the cost of overall speed. Performance also depends on the roughness of the terrain; hills and uneven surfaces hinder tracking performance.

Demonstrations of balancing and walking on surfaces that are not horizontal are shown in Figure 6.6. With the friction cone approximation given in Section 6.3.4, the robot is able to balance on a rotating platform until a 43-degree slope is reached. Narrower friction cone approximations are unable to reach the same tilt, as they are unable to provide the required tangential forces. In practice, this leads to the optimizer failing to converge when the desired COM wrench cannot be produced by the frictional point contacts. As for the walking, the controller is able to reliably ascend and descend inclined surfaces up to a 31-degree slope. The controller expects four contacts with the terrain on each foot. Fewer contacts, at the crests of hills for example, will produce jitter until four contacts are made. The end result is a robust walking behavior capable of executing motions resulting from our planning framework.

6.6 Motion Planning

For trajectory planning, we extended a three degree-of-freedom ARA* planning implementation to handle terrain with various slopes. Had we desired trajectories including foot locations, we could have built upon the methods developed in [48, 49]. The methods used in those papers, however, are only applicable to horizontal terrain.

If we utilized the same approach as planning for door opening from Chapter 5, we would ultimately get a full-dimensional plan including all joint angles for the legs and attempt to use position control to execute it. However, because the terrain we are operating on is only partially known and because falling would present an irrecoverable failure, some of the mapping to the full-dimensional plan was moved to the walking controller. Thus, the robot is likely to remain balanced despite uncertainty in contacts and ground reaction forces.

6.6.1 Graph Representation

The graph is constructed using a lattice-based representation, as described in Chapter 3. A lattice is a discretization of the configuration space into a set of states and connections between those states, where every connection represents a feasible path.

With respect to the framework of Chapter 4, let $\mathcal{X} \subset SE(3)$ represent the pelvis pose, $\mathcal{Y} \subset \mathbb{R}^n$ represent the set of leg configurations, $\mathcal{Z} \subset SE(3) \times SE(3)$ represent the poses of the feet, and $\mathcal{W} \in \{0, 1, 2\}$ represent whether the left foot, right foot, or both are in contact with the terrain. We emphasize that $\mathcal{Y}$ contains all valid leg configurations associated with poses chosen from $\mathcal{X}$ and $\mathcal{Z}$. Any state in the full-dimensional state-space, $s^f \in S^f$, can thus be represented by

$$\begin{aligned} s^f = (&\underbrace{x_p, y_p, z_p, \psi_p, \phi_p, \theta_p}_{\mathcal{X}}, \underbrace{q_{l_1}, \dots, q_{l_6}, q_{r_1}, \dots, q_{r_6}}_{\mathcal{Y}}, \\ &\underbrace{x_r, y_r, z_r, \psi_r, \phi_r, \theta_r, x_l, y_l, z_l, \psi_l, \phi_l, \theta_l}_{\mathcal{Z}}, \underbrace{k}_{\mathcal{W}}) \end{aligned}$$

where $(x, y, z, \psi, \phi, \theta)$ is a generic 6-degree-of-freedom pose, $\mathbf{q}_l$ and $\mathbf{q}_r$ are the leg joint angle vectors, and k represents the foot in contact. Thus, neglecting the joint angles for the upper body (we will assume these to be kept constant), we have a 31-degree-of-freedom state-space.

Again with respect to the framework of Chapter 4, and without loss of generality, let us consider the reduced-dimensional graph without the the superscript l for cleaner notation. Let $G = (S, T)$ denote the graph G we construct, with S the set of states and T the set of transitions between states. To discuss the states in S, let us first consider the motion of a humanoid walking and constraints we can apply to reduce the dimensionality of the space. Let $(x_p, y_p, \theta_p) \subset SE(2)$ represent the location and yaw of the pelvis. The height of the pelvis will handled by the controller, tracking a nominal height above the stance foot, and the roll and pitch are to be tracked are zero. If the terrain were flat, given the relative foot and pelvis positions, the joint angles for both 6-degree-of-freedom legs could be calculated using inverse kinematics; this is the abstraction of removing $\mathfrak{Y}$ from the lower-dimensional planning state. However, at plan time, our knowledge of distant terrain is inexact; we assume values for height and surface normals are noisy. Instead of using the relative locations of the feet to recover a full-dimensional plan for the lower half of the robot in advance, we use them to inform the search via the costmap as discussed in a following section, effectively removing $\mathfrak{Z}$ from our planning as well.

Accordingly, a state in the reduced-dimensional state-space used by the planner,

$s \in S$, is given by

$$s = (x_p, y_p, \theta_p) \subset SE(2)$$

for the planar location and yaw of the pelvis. The height of the pelvis is tracked by the controller, and the roll and pitch set to zero. One could consider successive levels of mapping back to higher-dimensional state-spaces. The first such mapping could be appending the relative footprint locations, with the next higher being the full joint states of the legs. The key difference between this application and that of opening doors is that for opening doors, the full-dimensional plan was first generated from the lower-dimensional plan and then executed verbatim. Here, rather than the mapping taking place in our planner, the planner is created such that the mapping is guaranteed to exist (i.e., infinite cost assigned to states where it would not exist), but the execution in and mapping to full-dimensional space is handled by the walking controller.

6.6.2 Implementation

As with opening doors, we use Anytime Repairing A* (ARA*) [66] to incrementally generate and search the graph. The algorithm generates an initial, possibly suboptimal solution then focuses on improving the solution while time remains. Here we provide additional details on the search setup.

Motion primitives

The environment has been discretized into 10 cm cells and the yaw into 22.5° bins. Thus, each primitive must be an integer multiple of 10 cm and 22.5°. Being kinematic motion primitives, all that is required is to interpolate between each start and goal location. Five primitives are used: 10 cm motion forward, an 80 cm motion forward while moving to the left 10 cm and turning left 22.5°, the preceding primitive turning to the right, and turning in place left or right. Each primitive is assigned an additional action cost multiplier; this is multiplied by the path length to get the total cost to execute the motion primitive in free space. The multipliers for the five primitives listed are: 1, 5, 5, 15, 15.

Costmap

We construct a 2-D costmap for the 3-D terrain using the following cost function:

$$c_{costmap} = \alpha|D| + \beta|dH|$$

where D is the slope of the terrain with respect to the horizontal and H is the height of the terrain the robot is standing on with respect to the height at the starting position. The reachable workspace of the legs (shown in Figure 6.7) was examined to determine traversable terrain height and gradient changes, which in turn informed the weights α and β. Areas above a certain height were also automatically classified as impassible obstacles, forcing the path to avoid the crests of hills. Sample costmaps for hilly terrain are shown in Figure 6.9.

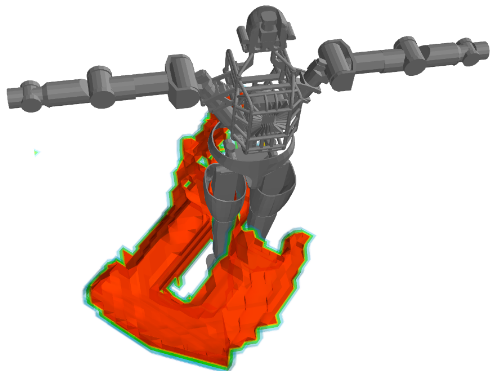


Figure 6.7: The reachable workspace for the Atlas's right foot is shown in orange.

Cost Function

The cost of a transition in our graph representation is defined as the product of two terms:

$$c(s, s') = c_{movement} \times (c_{costmap} + 1)$$

In the first term, $c_{movement}$ is the cost associated with moving the robot through a given base motion primitive. This term depends on the time it takes to execute the motion primitive. It also allows the user to minimize the use of certain primitives by assigning them higher costs. $c_{costmap}$ is given by convolving the robot footprint in the 2-D costmap as it executes the motion (the swept path). Because the motion primitives are precomputed, so are their paths. The relative set of cost map cells occupied by the robot during the motion are stored for each primitive. The occupied cells are also used to perform collision checking; an infeasible motion will have an infinite $c_{costmap}$.

We have calculated the reachable workspace of the Atlas's 6-degree-of-freedom legs using OpenRAVE [31]. Using this information, we precompute trajectories through the reachable workspace for new footsteps. We command the robot to execute the same open-loop lifting motion regardless of the coming terrain. The motion is always the same during leg raise, then deformed to place the foot above the new desired location at the same height as the stance foot. The pelvis begins to lower as the swing foot is lowered, allowing stepping down onto lower terrain while still succeeding on raised or level terrain. In this way, the stepping motion is robust to the upcoming terrain height. The leg raise motion, being known, is used to inform the costmap generation. Specifically, differences in terrain height that would cause collision lead to those regions being assigned infinite cost.

Heuristic

We use a simple heuristic, a 2-D Dijkstra search originating from the goal state, using the underlying grid at the resolution of the discretization. Because (many of) the motion primitives are larger than the discretization and may take any path, this will be an approximation of the remaining distance to the goal. It is worth noting that the motion primitive cost is a multiple of the Euclidean distance and thus the distance remains an underestimate of the cost.

6.6.3 Results

We used the ARA* planner to successfully generate COM trajectories then used the walking controller to follow them using a lookahead of 35 cm (approximately 1.5 step-lengths). Figure 6.8 shows that the robot is able track a desired trajectory to within 20 cm on each axis. The error is roughly sinusoidal, as follows from the pelvis oscillating back and forth across the trajectory when switching from foot to foot. Table 6.1 provides the planning times, expanded states, and relative solution costs for plans to the five goal locations shown in Figure 6.9c. For reference, the costmap is 40 cm by 16 m in size, containing 1,024,000 possible states. Initial solutions for larger values of ϵ are found quickly, in under 2 seconds, with the amount of time taken proportional to the number of expansions required to find the first solution, which is related to the distance to the goal and goal accessibility (not all primitives may make it through a narrow passageway). The final $\epsilon = 1$ solutions took up to 3 seconds, but since ARA* is an anytime method, the planner could have stopped anytime after the first solution if allowed time expired. For all the goals, the final solution after additional expansions is very near in cost or identical to the first solution, so truncating the search after the first solution would not have negatively impacted the resulting plans.

Goal	Planning Time (s)		Expansions		Path Cost	
	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$	First Soln. $\epsilon = 5.0$	Final Soln. $\epsilon = 1.0$
1	1.83	2.69	413,029	565,937	573,104	572,646
2	0.21	2.83	42,147	470,427	367,832	367,062
3	0.03	0.85	6,922	142,370	224,548	224,548
4	0.14	1.59	32,775	262,665	239,430	239,430
5	0.05	1.02	16,922	174,991	232,492	228,888

Table 6.1: Planning times, expanded states, and costs for paths through the hilly terrain. Goal locations are shown in Figure 6.9(c).

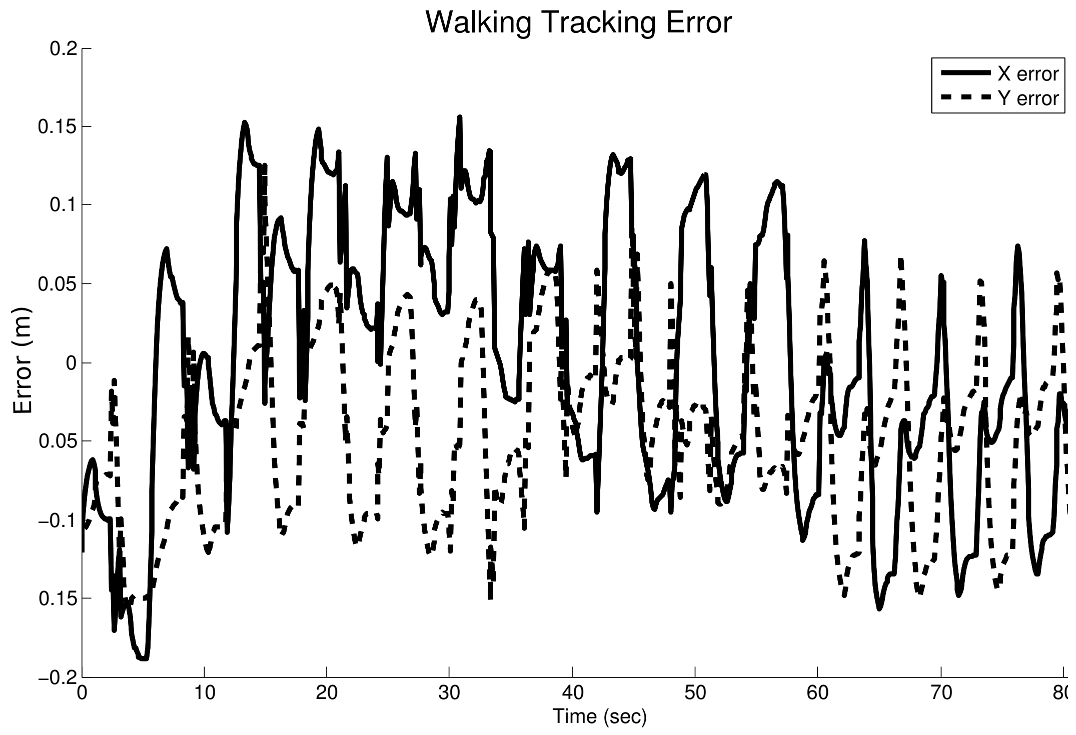
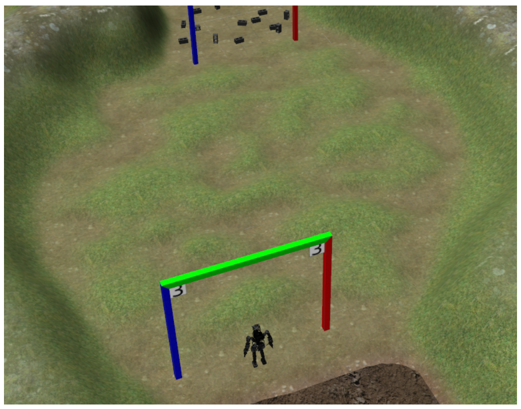
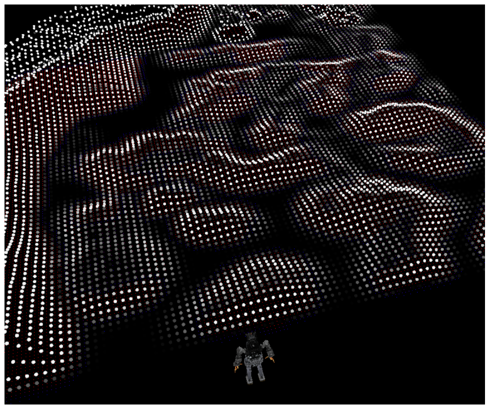


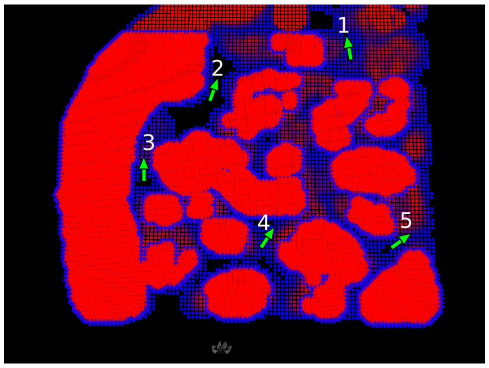
Figure 6.8: X,Y tracking error walking from the start position to the fourth goal location.



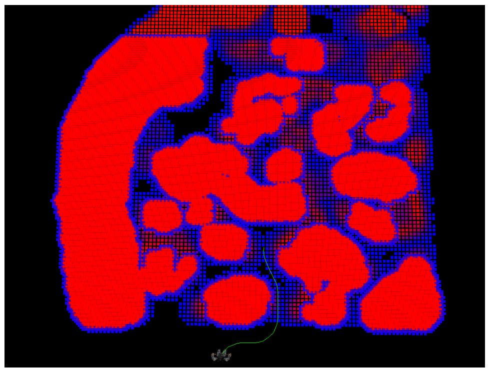
(a)



(b)



(c)



(d)

Figure 6.9: Stills from the planner operating in hilly terrain. (a) shows the Atlas robot standing before hills in the Gazebo simulation, (b) the voxelized representation of the world used to create the costmap, (c) the costmap given to the planner, with the goal locations used to generate Table 6.1, and (d) the costmap with a trajectory for the COM shown in green, leading to goal 4.

Chapter 7

Concluding Remarks

7.1 Summary of Contributions

Motion primitive-based (lattice-based) graphs have many desirable properties: they are optimal (or boundedly-suboptimal) for a given cost function and they return consistent, deterministic paths. However, their application to high-dimensional state-spaces has remained limited because of computational complexity. We have shown how to use lattice-based graphs in mobile manipulation with closed chains, exploiting the dimensionality reduction inherent in closed kinematic chains to enable efficient search.

This thesis contains two key contributions. First, to reduce complexity, we introduce abstractions for mobile manipulation with closed chains. The abstractions provide the dimensionality reduction to help make search-based planning tractable and also provide guarantees for generating an optimal full-state plan from the correspondingly optimal reduced-dimensional plan. The framework is structured such that transitions between contacts and changes between open and closed chains can be accommodated in a single planner. Second, this is the first time that motion plans for have been automatically generated by a single, unified planner for mobile manipulators performing such tasks as opening doors with switching contacts. These tasks include hybrid dynamics and constraints on forces/torques that can be applied. Our approach to door opening is validated by an extensive set of experiments performed using the PR2 robot, opening multiple doors (pulling and pushing both spring- and non-spring-loaded doors) in different buildings and universities in over 30 trials.

7.2 Future Work

During the course of working on this thesis, multiple challenges arose and were addressed, but warrant further study. One challenge is how to quickly generate code for each search space. Currently, each space has to be hand-coded, with the designer choosing which variables to include for the full and reduced-dimensional states. Another challenge is how to recognize modifications to the search space that lead to a more compact graph representation of the problem. For instance, when planning for doors, using a binary variable for the door interval rather than the multiple values a discretized door angle would require led to a dramatic reduction in planning times. Similarly, there remains the trade-off between including a wider variety of motion primitives and planning times. Smaller primitives may help the planner find paths in narrow passageways, while longer primitives may have faster planning times because fewer expansions are required. The best choice is often domain-dependent. Of course, including a large number of primitives, both long and short, negatively affects the time required for each expansion which is proportional to the total number of primitives. One promising avenue for automatically finding lower dimensional representations is the work of Vernaza *et al.* [105–107]. Vernaza's work focuses on identifying the low-dimensional Lagrangian structure of physical systems and applying this knowledge to aid in high-dimensional motion planning. The algorithm learns and exploits the structure of holonomic motion planning problems using spectral analysis and iterative dynamic programming and is able to solve problems with very high dimensions, with examples up to 990 dimensions.

The framework for combined open and closed-chain planning for manipulation presented here required the manipulator motions to be a path-connected set. In implementation, this required that, when confronted with possible disparate sets, the planner limited itself to one of them. This restriction can be eliminated by modifying the algorithm such that the planning is iterative: the entire path is first planned in the reduced-dimensional space, then reconstructed in the high-dimensional space piece-by-piece. If the higher-dimensional plan cannot be reconstructed at a given state, a higher-dimensional region is inserted into the low-dimensional representation. The low-dimensional planning is rerun, now with one or more higher-dimensional regions inside. This approach is similar to the adaptive dimensionality work of [38, 39].

One could also imagine applying this methodology to a problem which is continuous by nature. It has recently been suggested that we consider the problem of planning for a manipulator folding cloth. Creases may be inserted anywhere; there is no pre-defined crease pattern. The manipulator is limited to grabbing an edge of the cloth. To use our approach here, we must decide upon a discretization underlying the space. A natural choice for this is a regular pattern of creases, seen for instance in [43, 47], or perhaps a different pattern motivated by prior knowledge. According to our framework, we can think of $\mathcal{X}$ being the manipulator base location (most likely fixed), $\mathcal{Y}$ the set of manipulator arm configurations, $\mathcal{Z}$ the cloth crease state, and $\mathcal{W}$ encoding the specific contact locations (also discretized). Finding the valid transitions for the cloth crease state will involve combinatorial search, motivating a coarse crease pattern.

Lastly, a number of difficulties arise when attempting to execute planned paths in the real world. Findings in planning for uncertainty [8, 19] should be applied to this work, as well as further investigation into increasing robustness of plan execution. Specific difficulties for door opening included door localization, synchronized execution of base and arm motions, and difficulties in smoothing arm motions along with those of the base. Detection of the ARToolkit markers turned out to be a nontrivial problem; image thresholding parameters had to be adjusted based on light levels in different rooms. Even so, incorrect or slightly off detections resulted in the failure to grasp the door handle or poor grasps that were prone to slip. Additionally, the controller framework of the PR2 caused difficulties synchronizing the execution of arm and base motions. The trajectories generated for the PR2 include zero joint velocity for the arms after each motion. Even with smoothing, the motions are still noticeably jerky, but current inability to smooth the motion of the base and arms in a linked fashion limits the amount of smoothing possible. While we showed that the PR2 could be used to pull open spring-loaded doors up to roughly 20 N, spring-loaded doors often require around 60 N to open [51] and necessitate alternative approaches. For instance, the robot could pull open much heavier doors by extending the arm to a singular configuration so force at the end-effector depends on the strength of the base. However, the robot would then have to release the door and quickly move to make sure it passes through the door before it closes. This could be used in concert with dynamic motions, flinging the door open and immediately moving through. Specific difficulties for the simulated humanoid included dealing with unknown terrain and tracking exact footstep locations without knowledge of ground truth.

Extending the framework and improving execution robustness will provide interesting and immediate benefits for the work in this thesis. Further study of motion primitive-based planning will undoubtedly make design choices easier for those crafting their own searches. My work has established a framework for using lattice-based graphs in mobile manipulation with closed chains, which will eventually lead to robots navigating our buildings and opening doors, moving furniture, and otherwise interacting with human environments.

Bibliography

[1] A. E. Leeper A. T. Pratkanis and J. K. Salisbury, *Replacing the office intern: An autonomous coffee run with a mobile manipulator*, IEEE International Conference on Robotics and Automation, 2013.

[2] Saleh Ahmad and Guangjun Liu, *A door opening method by modular reconfigurable robot with joints working on passive and active modes*, IEEE International Conference on Robotics and Automation, 2010, pp. 1480–1485.

[3] D. Anguelov, D. Koller, E. Parker, and S. Thrun, *Detecting and modeling doors with mobile robots*, IEEE International Conference on Robotics and Automation, 2004.

[4] Hitoshi Arisumi, Jean-Rémy Chardonnet, and Kazuhito Yokoi, *Whole-body motion of a humanoid robot for passing through a door*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 428–434.

[5] Eliana P. L. Aude, Ernesto P. Lopes, Cristiano S. Aguiar, and Mario F. Martins, *Door crossing and state identification using robotic vision*, 8th International

IFAC Symposium on Robot Control, September 2006.

[6] D. Berenson, T. Simeon, and S.S. Srinivasa, *Addressing cost-space chasms in manipulation planning*, IEEE International Conference on Robotics and Automation, May 2011, pp. 4561–4568.

[7] Dmitry Berenson and Siddhartha Srinivasa, *Probabilistically complete planning with end-effector pose constraints*, IEEE International Conference on Robotics and Automation, 2010.

[8] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner, *Addressing pose uncertainty in manipulation planning using task space regions*, IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2009.

[9] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner, *Task space regions: A framework for pose-constrained manipulation planning*, International Journal of Robotics Research (2011).

[10] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, and James J. Kuffner, *Manipulation planning on constraint manifolds*, IEEE International Conference on Robotics and Automation, May 2009, pp. 625–632.

[11] R. Bohlin and L.E. Kavraki, *Path planning using lazy PRM*, IEEE International Conference on Robotics and Automation, vol. 1, 2000, pp. 521–528.

[12] Boston Dynamics Incorporated, *Atlas The Agile Anthropomorphic Robot*, 2013, http://www.bostondynamics.com/robot_Atlas.html.

[13] Michael S Branicky, Ross A Knepper, and James J Kuffner, *Path and trajectory diversity: Theory and algorithms*, IEEE International Conference on Robotics and Automation, pp. 1359–1364.

[14] Timothy Bretl and Sanjay Lall, *Testing static equilibrium for legged robots*, IEEE Transactions on Robotics **24** (2008), no. 4, 794–807.

[15] O. Brock and O. Khatib, *High-speed navigation using the global dynamic window approach*, IEEE International Conference on Robotics and Automation, 1999, pp. 341–346.

[16] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, *Six-dof impedance control based on angle/axis representations*, IEEE Transactions on Robotics and Automation **15** (1999), no. 2, 289–300.

[17] J.F. Canny, *The complexity of robot motion planning*, MIT Press, Inc., 1988.

[18] W.F. Carriker, P.K. Khosla, and B.H. Krogh, *An approach for coordinating mobility and manipulation*, IEEE International Conference on Systems Engineering, 1989, pp. 59–63.

[19] Andrea Censi, Daniele Calisi, Alessandro De Luca, and Giuseppe Oriolo, *A Bayesian framework for optimal motion planning with uncertainty*, IEEE International Conference on Robotics and Automation, May 2008.

[20] Lillian Y Chang, Siddhartha S Srinivasa, and Nancy S Pollard, *Planning pre-grasp manipulation for transport tasks*, IEEE International Conference on Robotics and Automation, 2010, pp. 2697–2704.

[21] Sachin Chitta, Benjamin Cohen, and Maxim Likhachev, *Planning for autonomous door opening with a mobile manipulator*, IEEE International Conference on Robotics and Automation, 2010.

[22] Matei Ciocarlie, Kaijen Hsiao, E Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan A Sucan, *Towards reliable grasping and manipulation in household environments*, International Symposium on Experimental Robotics, 2010, pp. 1–12.

[23] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev, *Search-based planning for manipulation with motion primitives*, IEEE International Conference on Robotics and Automation, 2010.

[24] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev, *Search-based planning for dual-arm manipulation with upright orientation constraints*, IEEE International Conference on Robotics and Automation, 2012.

[25] Benjamin Cohen, Gokul Subramanian, Sachin Chitta, and Maxim Likhachev, *Planning for manipulation with adaptive motion primitives*, IEEE International Conference on Robotics and Automation, 2011.

[26] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki, *The open motion planning library*, IEEE Robotics & Automation Magazine **19** (2012), no. 4, 72–82.

[27] Sébastien Dalibard, Alireza Nakhaei, Florent Lamiraux, and Jean-Paul Laumond, *Manipulation of documented objects by a walking humanoid robot*, IEEE-RAS International Conference on Humanoid Robots, 2010, pp. 518–523.

[28] T. L. Dean and M. Boddy, *An analysis of time-dependent planning*, National Conference on Artifical Intelligence, 1988.

[29] Defense Advanced Research Projects Agency, *The DARPA Robotics Challenge*, 2013, http://theroboticschallenge.org/.

[30] R. Diankov, S. Srinivasa, D. Ferguson, and J. Kuffner, *Manipulation planning with caging grasps*, IEEE-RAS International Conference on Humanoid Robots, December 2008.

[31] Rosen Diankov, *Automated construction of robotic manipulation programs*, Ph.D. thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[32] G. Digioia, H. Arisumi, and K. Yokoi, *Trajectory planner for a humanoid robot passing through a door*, IEEE-RAS International Conference on Humanoid Robots, December 2009, pp. 134–141.

[33] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, no. 1, 1959, pp. 269–271.

[34] G. Dumonteil, *ARToolKit package for ROS*, August 2011, www.ros.org/wiki/artoolkit.

[35] N. Egerstedt and Xiaoming Hu, *Coordinated trajectory following for mobile manipulation*, IEEE International Conference on Robotics and Automation, vol. 4, 2000, pp. 3479–3484.

[36] Lawrence H Erickson and Steven M LaValle, *Survivability: Measuring and ensuring path diversity*, IEEE International Conference on Robotics and Automation, IEEE, 2009, pp. 2068–2073.

[37] RIKEN-TRI Collaboration Center for Human-Interactive Robot Research, *RIBA robot for interactive body assistance*, 2013, http://rtc.nagoya.riken.jp/RIBA/index-e.html.

[38] Kalin Gochev, Benjamin Cohen, Jonathan Butzke, Alla Safanova, and Maxim Likhachev, *Path planning with adaptive dimensionality*, Fourth International Symposium on Combinatorial Search, 2011.

[39] Kalin Gochev, Alla Safonova, and Maxim Likhachev, *Planning with adaptive dimensionality for mobile manipulation*, IEEE International Conference on Robotics and Automation, 2012.

[40] Steven Gray, Sachin Chitta, Vijay Kumar, and Maxim Likhachev, *A single planner for a composite task of approaching, opening and navigating through non-spring and spring-loaded doors*, IEEE International Conference on Robotics and Automation, 2013.

[41] Steven Gray, Christopher Clingerman, Sachin Chitta, Vijay Kumar, and Maxim Likhachev, *Search-based planning for autonomous spring-loaded door opening*, Robotics: Science and Systems (RSS) Workshop on Mobile Manipulation: Generating Robot Motion for Contact with the World, 2012.

[42] Steven Gray, Christopher Clingerman, Sachin Chitta, and Maxim Likhachev, *PR2: Opening spring-loaded doors*, IEEE-RSJ International Conference on Intelligent Robots and Systems, PR2 Workshop, 2011.

[43] Steven Gray, Nathan Zeichner, Vijay Kumar, and Mark Yim, *A simulator for origami-inspired self-reconfigurable robots*, Origami 5: Fifth International Meeting of Origami Science, Mathematics, and Education, CRC Press, 2011, p. 323.

[44] Colin J Green and Alonzo Kelly, *Toward optimal sampling in the space of paths*, Robotics Research, Springer, 2011, pp. 281–292.

[45] M Hans, B Graf, and RD Schraft, *Robotic home assistant care-o-bot: Past-present-future*, IEEE International Workshop on Robot and Human Interactive Communication, 2002, pp. 380–385.

[46] 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** (1968), no. 2, 100–107.

[47] Elliot Hawkes, B An, NM Benbernou, H Tanaka, S Kim, ED Demaine, D Rus, and RJ Wood, *Programmable matter by folding*, Proceedings of the National Academy of Sciences **107** (2010), no. 28, 12441–12445.

[48] Armin Hornung, Andrew Dornbush, Maxim Likhachev, and Maren Bennewitz, *Anytime search-based footstep planning with suboptimality bounds*, IEEE-RAS International Conference on Humanoid Robots, November 2012.

[49] Armin Hornung, Daniel Maier, and Maren Bennewitz, *Search-based footstep planning*, IEEE International Conference on Robotics and Automation: Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids (Karlsruhe, Germany), May 2013.

[50] Sang-Ho Hyon, Joshua G Hale, and Gordon Cheng, *Full-body compliant human–humanoid interaction: Balancing in the presence of unknown external forces*, IEEE Transactions on Robotics **23** (2007), no. 5, 884–898.

[51] A. Jain, Hai Nguyen, M. Rath, J. Okerman, and C. C. Kemp, *The complex structure of simple devices: A survey of trajectories and forces that open doors and drawers*, IEEE-RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics, September 2010, pp. 184–190.

[52] Advait Jain and Charles C. Kemp, *Pulling open doors and drawers: Coordinating an omni-directional base and a compliant arm with equilibrium point control*, IEEE International Conference on Robotics and Automation, 2010, pp. 1807–1814.

[53] Stefan Jörg, Jörg Langwald, Ciro Natale, Johannes Stelter, and Gerd Hirzinger, *Flexible robot-assembly using a multi-sensory approach*, IEEE International Conference of Robotics and Automation, 2000, pp. 3687–3694.

[54] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, *Stomp: Stochastic trajectory optimization for motion planning*, IEEE International Conference on Robotics and Automation, May 2011, pp. 4569–4574.

[55] Ju-Hyun Kang, Chang-Soon Hwang, and Gwi Tae Park, *A simple control method for opening a door with mobile manipulator*, International Conference on Control, Automation and Systems, 2003.

[56] S. Karaman and E. Frazzoli, *Incremental sampling-based optimal motion planning*, Robotics: Science and Systems, 2010.

[57] L.E. 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** (1996), no. 4, 566–580.

[58] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, International Journal of Robotics Research **5** (1986), no. 1, 90–98.

[59] E. Klingbeil, A. Saxena, and A. Y. Ng, *Learning to open new doors*, Robotics: Science and Systems Workshop on Robot Manipulation: Intelligence in Human Environments, 2008.

[60] Vijay Kumar, *Instantaneous kinematics of parallel-chain robotic mechanisms*, Journal of Mechanical Design **114** (1992), 349.

[61] Aleksandr Kushleyev, Brian MacAllister, and Maxim Likhachev, *Planning for landing site selection in the aerial supply delivery*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Sept. 2011, pp. 1146–1153.

[62] J-C. Latombe, *Robot motion planning*, Kluwer Academic Publishers, 1991.

[63] S.M. LaValle, M.S. Branicky, and S.R. Lindemann, *On the relationship between classical grid search and probabilistic roadmaps*, International Journal of Robotics Research (2003).

[64] Steven M Lavalle, *Rapidly-exploring random trees: A new tool for path planning*, Tech. Report 98-11, Computer Science Department, Iowa State University, October 1998.

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

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

[67] G.F. Liu, J.C. Trinkle, and R.J. Milgram, *Complete path planning for a planar 2-R manipulator with point obstacles*, IEEE International Conference on Robotics and Automation, vol. 4, May, 2004, pp. 3263–3269.

[68] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev, *Path planning for non-circular micro aerial vehicles in constrained environments*, IEEE International Conference on Robotics and Automation, 2013.

[69] A. Madhani and S. Dubowsky, *Motion planning of mobile multi-limb robotic systems subject to force and friction constraints*, IEEE International Conference on Robotics and Automation, May 1992, pp. 233–239.

[70] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, *The office marathon: Robust navigation in an indoor office environment*, IEEE International Conference on Robotics and Automation, 2010.

[71] J.E. Mattingley, S.P. Boyd, M.A. Saunders, Y. Ye, and Stanford University. Dept. of Electrical Engineering, *Code generation for embedded convex optimization*, Stanford University, 2011.

[72] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger, *Autonomous door opening and plugging in using a personal robot*, IEEE International Conference on Robotics and Automation, 2010.

[73] Richard M. Murray, S. Shankar Sastry, and Li Zexiang, *A mathematical introduction to robotic manipulation*, 1st ed., CRC Press, Inc., Boca Raton, FL, USA, 1994.

[74] K. Nagatani and S. Yuta, *An experiment on opening-door-behavior by an autonomous mobile robot with a manipulator*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 1995, pp. 45–50.

[75] Shigeki Nakaura and Mitsuji Sampei, *Balance control analysis of humanoid robot based on ZMP feedback control*, IEEE-RSJ International Conference on Intelligent Robots and Systems, vol. 3, 2002, pp. 2437–2442.

[76] G. Niemeyer and J. Slotine, *A simple strategy for opening an unknown door*, IEEE International Conference on Robotics and Automation, 1997.

[77] Guiseppe Oriolo and Christian Mongillo, *Motion planning for mobile manipulators along given end-effector paths*, IEEE International Conference on Robotics and Automation, 2005.

[78] C. Ott, M.A. Roa, and G. Hirzinger, *Posture and balance control for biped robots based on contact force optimization*, IEEE-RAS International Conference on Humanoid Robots, 2011, pp. 26–33.

[79] Christian Ott, Berthold Bäuml, Christoph Borst, and Gerd Hirzinger, *Employing cartesian impedance control for the opening of a door: A case study in mobile manipulation*, IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop on Mobile Manipulators: Basic Techniques, New Trends & Applications, 2005.

[80] Christian Ott, Berthold Bäuml, Christoph Borst, and Gerd Hirzinger, *Autonomous opening of a door with a mobile manipulator: A case study*, IFAC Symposium on Intelligent Autonomous Vehicles, 2007.

[81] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi, *Motion planning through symbols and lattices*, IEEE International Conference on Robotics and Automation, vol. 4, 2004, pp. 3914–3919.

[82] J. Pearl, *Heuristics: Intelligent search strategies for computer problem solving*, Addison-Wesley, 1984.

[83] Alejandro Perez, Sertac Karaman, Alexander Shkolnik, Emilio Frazzoli, Seth Teller, and Matthew R. Walter, *Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Sept. 2011, pp. 4307–4313.

[84] L. Petersson, D. Austin, and D. Kragic, *High-level control of a mobile manipulator for door opening*, IEEE-RSJ International Conference on Intelligent Robots and Systems, vol. 3, 2000, pp. 2333–2338.

[85] R. Philippsen and R. Siegwart, *Smooth and efficient obstacle avoidance for a tour guide robot*, IEEE International Conference on Robotics and Automation, 2003, pp. 446–451.

[86] Mike Phillips, Benjamin Cohen, Sachin Chitta, and Maxim Likhachev, *E-graphs: Bootstrapping planning with experience graphs*, Robotics: Science and Systems, 2012.

[87] Mike Phillips, Andrew Dornbush, Sachin Chitta, and Maxim Likhachev, *Anytime incremental planning with E-graphs*, IEEE International Conference on Robotics and Automation, 2013.

[88] M. Pivtoraiko and A. Kelly, *Generating near minimal spanning control sets for constrained motion planning in discrete state spaces*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Aug. 2005, pp. 3231–3237.

[89] Mihail Pivtoraiko and Alonzo Kelly, *Kinodynamic motion planning with state lattice motion primitives*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Sept. 2011, pp. 2172–2179.

[90] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar, *Quadrotor maneuver generation using motion primitives*, IEEE International Conference on Robotics and Automation, 2013.

[91] José P. Puga and Luciano E. Chiang, *Optimal trajectory planning for a redundant mobile manipulator with non-holonomic constraints performing push-pull tasks*, Robotica **26** (2008), 385–394.

[92] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa, *Chomp: Gradient optimization techniques for efficient motion planning*, IEEE International Conference on Robotics and Automation, May 2009, pp. 489–494.

[93] Ulrich Reiser, Christian Connette, Jan Fischer, Jens Kubacki, Alexander Bubeck, Florian Weisshardt, Theo Jacobs, Christopher Parlitz, M Hagele, and Alexander Verl, *Care-o-bot® 3-creating a product vision for service robot applications by integrating design and technology*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 1992–1998.

[94] Rethink Robotics, *Baxter: A Unique Robot with Unique Features*, 2013, http://www.rethinkrobotics.com/index.php/products/baxter.

[95] C. Rhee, W. Chung, M. Kim, Y. Shim, and H. Lee, *Door opening control using the multi-fingered robotic hand for the indoor service robot*, IEEE International Conference on Robotics and Automation, 2004.

[96] S. Rodriguez, Xinyu Tang, J. M. Lien, and Nancy M. Amato, *An obstacle-based rapidly-exploring random tree*, IEEE International Conference on Robotics and Automation, 2006.

[97] T. Ruhr, J. Sturm, D. Pangercic, M. Beetz, and D. Cremers, *A generalized framework for opening doors and drawers in kitchen environments*, IEEE International Conference on Robotics and Automation, 2012.

[98] John Schulman, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel, *Finding locally optimal, collision-free trajectories with sequential convex optimization*, Robotics: Science and Systems, 2013.

[99] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani, *Manipulation planning with probabilistic roadmaps*, The International Journal of Robotics Research **23** (2004), no. 7-8, 729–746.

[100] Tomomichi Sugihara and Yoshihiko Nakamura, *Whole-body cooperative balancing of humanoid robot using cog jacobian*, IEEE-RSJ International Conference on Intelligent Robots and Systems, vol. 3, 2002, pp. 2575–2580.

[101] Xinyu Tang, S. Thomas, and N.M. Amato, *Planning with reachable distances: Fast enforcement of closure constraints*, IEEE International Conference on Robotics and Automation, April 2007, pp. 2694–2699.

[102] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt, *Map learning and high-speed navigation in RHINO*, AI-based Mobile Robots: Case Studies of Successful Robot Systems (D. Kortenkamp, R.P. Bonasso, and R Murphy, eds.), MIT Press, 1998.

[103] J.C. Trinkle and R.J. Milgram, *Motion planning for planar n-bar mechanisms with revolute joints*, IEEE-RSJ International Conference on Intelligent Robots and Systems, vol. 3, 2001, pp. 1602–1608.

[104] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, *Humanoid motion planning for dual-arm manipulation and re-grasping tasks*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Oct. 2009, pp. 2464–2470.

[105] Paul Vernaza and Daniel D Lee, *Learning dimensional descent for optimal motion planning in high-dimensional spaces*, AAAI, 2011.

[106] Paul Vernaza and Daniel D Lee, *Learning and exploiting low-dimensional structure for efficient holonomic motion planning in high-dimensional spaces*, The International Journal of Robotics Research **31** (2012), no. 14, 1739–1760.

[107] Paul Vernaza, Daniel D Lee, and Seung-Joon Yi, *Learning and planning high-dimensional physical trajectories via structured Lagrangians*, IEEE International Conference on Robotics and Automation, 2010, pp. 846–852.

[108] Miomir Vukobratović and Branislav Borovac, *Zero-moment pointthirty five years of its life*, International Journal of Humanoid Robotics **1** (2004), no. 01, 157–173.

[109] B. J. W. Waarsing, M. Nuttin, and H. Van Brussel, *Behaviour-based mobile manipulation: The opening of a door*, International Workshop on Advances in Service Robotics, 2003, pp. 168–175.

[110] Glenn Wagner and Howie Choset, *M*: A complete multirobot path planning algorithm with performance bounds*, IEEE-RSJ International Conference on Intelligent Robots and Systems, Sept. 2011, pp. 3260–3267.

[111] Jason Wolfe, Bhaskara Marthi, and Stuart Russell, *Combined task and motion planning for mobile manipulation*, International Conference on Automated Planning and Scheduling, 2010, pp. 254–258.

[112] Dawen Xie and N.M. Amato, *A kinematics-based probabilistic roadmap method for high DOF closed chain systems*, IEEE International Conference on Robotics and Automation, vol. 1, May 2004, pp. 473–478.

[113] J.H. Yakey, S.M. LaValle, and L.E. Kavraki, *Randomized path planning for linkages with closed kinematic chains*, IEEE Transactions on Robotics and Automation **17** (2001), no. 6, 951–958.

[114] F. Zacharias, W. Sepp, C. Borst, and G. Hirzinger, *Using a model of the reachable workspace to position mobile manipulators for 3-D trajectories*, IEEE-RAS International Conference on Humanoid Robots, Dec. 2009, pp. 55–61.

[115] Rong Zhou and Eric A Hansen, *Multiple sequence alignment using Anytime A**, Proceedings of the National Conference on Artificial Intelligence, 2002, pp. 975–977.