MOTION PLANNING FOR LEGGED AND HUMANOID ROBOTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kris Hauser

December 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Jean-Claude Latombe) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Oussama Khatib)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Andrew Ng)

Approved for the University Committee on Graduate Studies.

Abstract

Legged vehicles have attracted interest for many high-mobility applications, such as military troop support and logistics in rocky, steep, and forested terrain, scientific exploration of cliffs, mountains, and volcanoes on earth and other planets, and search and rescue. Humanoid robots have additional applications in homes and offices as personal assistants. But planning and controlling their motion is difficult, because dozens of joints must be coordinated to maintain balance while executing a task. It is particularly difficult in extremely uneven, steep, or cluttered terrain — precisely where legs are an advantage.

I present a legged locomotion planner that reasons with the physical and operational constraints in the vehicle's high dimensional configuration space, producing motions that are guaranteed to avoid collision and remain balanced. This planner works on a variety of terrain, ranging from flat ground to steep and rocky cliff faces, and is applied to three very different robots: NASA's six-legged lunar vehicle ATHLETE, AIST Japan's biped humanoid robot HRP-2, and Stanford's four-limbed rock climbing robot Capuchin.

The planner consists of two parts: first, a graph search to find a sequence of contacts to make and break, and second, the planning of single-step motions between successive contacts using a probabilistic roadmap planner (PRM). I prove that a "fuzzy search", where PRM planning is interleaved between candidate steps, has theoretical completeness properties, and a running time not strongly affected by configuration space dimension. The planner can also produce natural-looking motions given a small number of "motion primitives", high-quality example motions that bias

the search toward similar paths. These techniques can be applied to other robot systems that make and break contact, and I present a manipulation planner that enables the Honda ASIMO humanoid robot to push objects on a table, in simulation and on the real robot.

Acknowledgements

I would first like to thank my advisor, Jean-Claude Latombe, for his inexhaustible wisdom. He has given me the guidance to be a better researcher, thinker, speaker, writer, and teacher, and for this I offer my sincerest appreciation.

This dissertation could not have been possible without the pioneering legged robot work of Tim Bretl. We have worked closely together on several projects, and he has always brought energy, insights, and new perspectives to all that he has worked on. I am indebted to my external collaborators and hosts, Kensuke Harada, Hirohisa Hirukawa, Brian Wilcox, Victor Ng-Thow-Hing, and Hector Gonzalez-Banos, who have inspired me with incredible opportunities to work with state-of-the-art robots.

Thanks to Latombe lab members Ruixiang Zhang, Peggy Yao, Philip Fong, Ankur Dhanik, and Mitul Saha the invaluable discussions we had throughout our studies. Thanks to Anita, Rupa, Miguel, Uzair, Monica, Rishi, Jason, Gaurav, Marsha, Nicole, Bronwen, Oliver, John Brian, Nathan, and Michael for your friendship, which helped me maintain some semblance of a personal life.

Many thanks to my parents for their lifelong support. Special thanks to my grandmother, who is a fountain of love, grace, and tenacity even at 95 years old. I also appreciate the (stereotype-defying) love and support from my in-laws. Thanks to little brothers Alan and Richard, and SFAM Maryn.

My deepest thanks go out to my wife, Ashlyn, whose emotional and physical support has given me strength and joy during the writing of this dissertation. She is the love of my life, and has been my world ever since we met.

I will end with a baby-sized thank you to my month-old daughter Saya, who will hopefully be able to understand this thesis some day.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Legged robots have been proposed for use in planetary exploration [58, 79], exploration of cliffs and volcanoes [7, 60], search and rescue [4], transportation in rough terrain [100], and building inspection [24], among other tasks. Legged vehicles are potentially better than wheeled ones in navigating steep, cluttered, or varying terrain because obstacles can be stepped over, stepped on, or avoided by reconfiguring the body. But these benefits have not yet been demonstrated convincingly in real-world systems, due in large part to the difficulty of planning and controlling the robot's motion [4]. Many joints (often a few dozen) must be coordinated to achieve high-level goals while maintaining stability and avoiding obstacles.

Motion planning is the process of computing robot motions automatically. This is a necessary capability for legged robots that are designed for full or partial autonomy. Motion planning is useful even if a robot is human-operated, as it can be difficult and painfully slow for a human to directly control a mechanism with many joints and complex motion constraints [74, 100]. A possible human-robot interface may consist of the operator issuing simple short term goals, with a motion planner automatically computing motions that achieve those goals [68, 119]. Motion planning could also facilitate robot mechanism design, by testing designs in simulation to verify their capabilities.

Legged robots move in a configuration space whose structure is unlike that of more "traditional" robots, like fixed base manipulators or wheeled robots. When a legged

robot contacts the environment at a fixed set of footfalls, the motion space lies in a submanifold of configuration space with certain motion constraints. If a footfall is broken, a kinematic constraint is removed, and the motion space switches to a higher-dimensional submanifold. If a new footfall is reached, additional constraints change the motion space to a lower-dimensional submanifold. Thus, a planner must produce a continuous motion through a discrete sequence of *modes*, where each mode is a fixed set of contacts with its own set of constraints. In control theory terminology, these are hybrid systems because they mix discrete and continuous behavior. The term *multi-modal planning* describes the general process of planning for hybrid systems.

The need for multi-modal planning arises in part manipulation with multiple grasp and regrasp operations [2, 43, 101, 112, 128], dexterous manipulation [35], legged locomotion [14, 22, 36, 57, 58, 81, 107], navigation among movable obstacles [102, 118], assembly planning [123], and reconfigurable robots [32, 33]. This thesis primarily focuses on locomotion planning for legged robots, particularly in steep and rough terrain, with a secondary focus on manipulation planning for humanoid robots. Humanoid robots are of interest for their potential applications as household, office, hospital, or hospice assistants, but pose an additional challenge for planning: they must be able to both navigate cluttered environments *and* manipulate objects.

1.1 Overview of Thesis

The three major components of this thesis address a broad range of issues in multi-modal planning:

- *Analysis of multi-modal planning.* The first component models, surveys, and analyzes algorithms for multi-modal motion planning in a problem-independent setting (Chapter 2), and describes an implementation of a legged locomotion planner for steep and irregular terrain (Chapter 3).

- *Planning strategies that use domain knowledge.* The second component describes two techniques that use domain knowledge (advance knowledge of the tasks the robot is expected to perform) to improve motion quality and planning

speed, one in legged locomotion on uneven terrain and the other in a manipulation planner for a humanoid robot (Chapter 4).

- *Decision theoretic planner optimization.* The final component takes a decision theoretic approach to optimizing the planner's high-level choices, providing a principled way to order subproblems and tune algorithm parameters for a wide variety of planning subroutines (Chapter 5).

These components are introduced in the following three sections.

1.2 Multi-Modal Planning

Most work in multi-modal planning has focused on developing planners for specific hybrid systems. Chapter 2 compares prior work in a problem-independent manner. It also describes a general-purpose multi-modal planning algorithm that provably satisfies a completeness result. Chapter 3 describes, in detail, an implementation of this algorithm in a legged locomotion planner.

A hybrid system travels between discrete modes, where each mode has its own feasibility constraints. A multi-modal planner must choose a *discrete* sequence of modes, as well as a *continuous* motion to traverse them. This may require solving hard problems in both the discrete and continuous domains. Motion planning in high-dimensional continuous spaces is hard; the worst case time of any complete algorithm to plan a collision-free motion of an articulated robot is exponential in the number of degrees of freedom [110]. But multi-modal problems may be hard even if each continuous space has low dimensionality; for example, navigation among moving obstacles in 2D is NP-hard when the number of obstacles is not fixed [122]. And yet, a few multi-modal planners have been developed for specific problems with some success.

Probabilistic roadmap (PRM) planners [71] are state-of-the-art motion planners for high-dimensional configuration spaces with complex geometric constraints. They build a network that approximates the connectivity of the set of feasible configurations, by randomly sampling points in the set and connecting them with straight-line

paths in configuration space. They only work within a single mode, but nevertheless are a key tool for multi-modal planning. To connect two modes (a start configuration in one mode to a goal configuration in another), we may pick an intermediate transition configuration that is feasible in both modes, and solve two single-mode PRM queries (one connecting the start to the transition, and another connecting the transition to the goal). This basic process is used widely in multi-modal planning, e.g., for manipulation planning [101, 112] and climbing robots [19]. But PRMs cannot report in finite time that no path exists. This presents a problem when a large number (thousands, or more) of single-mode planning queries must be answered, and a large fraction are infeasible. How long should a PRM try to solve a query before reporting failure? Too long, and the multi-modal planner is extremely slow. Too little, and the planner will miss many feasible paths.

A new algorithm in Chapter 2, Multi-Modal-PRM, handles this dilemma by building PRMs in parallel across modes. I prove that, given certain assumptions, the probability this algorithm fails to find a path, when one exists, approaches 0 exponentially as running time increases. Its running time, like that of PRMs, is mainly dependent on the shape properties of the feasible sets, and not their dimensionality.

However, even when a problem has an enormous number of modes (millions or more), relatively few mode switches are needed to answer a query (typically tens or hundreds). Using this observation allows us to speed up planning by orders of magnitude. An incremental variant of Multi-Modal-PRM restricts PRM planning to a small subset of candidate modes. Until a path to the goal is found, the planner alternates between growing the set of candidate modes and planning within them. This works well if, early on, we consider candidate modes that contain a path to the goal. To help do so, we use a "fuzzy" search that produces a sequence of feasible modes leading to the goal (similar to [22]).

Chapter 3 uses this algorithm in a multi-modal planner for legged locomotion, and describes the implementation of planning subroutines in detail. This planner was originally based on work by Tim Bretl on a planar rock-climbing robot [19], but makes several new contributions that enable it to plan tractably for arbitrary 3D robots in arbitrary terrain. In particular, it makes contributions in system modeling,

motion constraint testing, fast configuration sampling, routines to prune unnecessary modes, and search heuristics. This planner has been applied to several robots in irregular and steep terrain. In particular, it has been tested in simulation on NASA's ATHLETE six-legged lunar robot [58] and the HRP-2 humanoid developed by AIST Japan [70], and on the physical Capuchin rock-climbing robot developed by Ruixiang Zhang at Stanford [129]. This planner can plan motions in terrains where gaits fail, and even outperforms human operators on ATHLETE in difficult terrain. It has also discovered that HRP-2 can climb a large stair using its hands to aid stability, a previously unknown capability that was verified in a quasi-static simulator.

1.3 Using Domain Knowledge in Planning

A drawback of MULTI-MODAL-PRM is that it is difficult to produce natural-looking motions; not only do the single-mode motions need to look natural, but the mode selections must be natural as well. This problem is perhaps most apparent for humanoid robots and digital characters, because human observers can be distracted by even slight deviations from natural motion. An unnatural, jerky path can be difficult to repair in postprocessing: variational techniques can optimize the continuous path, but not discrete mode selections. Optimizing during planning is far too expensive. But some motions (e.g., taking a step on flat ground) are executed repeatedly; perhaps the planner could take advantage of the similarity between steps.

This idea is used in several alternative approaches. The robot could be restricted to repeating a small set of primitive actions (e.g., engineered gaits, motion clips) [36, 82, 90, 120]. Another approach is to first plan an overall motion of the robot's body (in locomotion planning) or the object (in manipulation planning), and try to follow the path using appropriate choices of contact [41, 76, 107, 121] or irregularity-tolerant controllers [25]. All these methods use *domain knowledge* to reduce the space of allowable motions. This typically has the effect of speeding up the planner and improving motion quality for a limited range of problems, at the expense of a higher failure rate on problems outside this range. This causes difficulties for non-repetitive tasks, such as navigating in rough and irregular environments where no two steps are

the same, as well as manipulation of irregular objects.

Chapter 4 presents two new techniques that, like the above approaches, use domain knowledge to improve planning quality and speed. But instead of reducing the space of allowed motions, domain knowledge is used to *bias* the planner's exploration toward promising choices. The planner uses high-quality motions when possible, but does not give up if this fails - it will resort to using low-quality motions if necessary.

The first technique uses *motion primitives*, high-quality example motions, to produce natural-looking paths for the HRP-2 humanoid on irregular terrain. Experiments in simulation show HRP-2 traversing varied terrain with natural-looking motions. The second technique uses a *utility precomputation* to improve a nonprehensile (pushing) manipulation planner for the Honda ASIMO humanoid robot. The utility measures the expected distance that the object can be pushed, and helps the planner choose long-distance pushes. Tests in simulation and experiments on the actual ASIMO robot show that long-range plans with of dozens of pushes can be generated in minutes.

1.4 Decision-Theoretic Planning

The designer of a multi-modal planner is faced with problems of *algorithm selection and tuning*, and *subproblem ordering*. For every problem, the designer must choose an algorithm and its parameters from possibly many alternatives. Similarly, when the problem requires solving many subproblems, the designer must choose an execution strategy. Due to the widespread use of randomized subroutines, this requires reasoning in the face of uncertainty. Even deterministic subroutines that solve complex problems may behave unpredictably.

Chapter 5 presents a decision-theoretic approach to reasoning about computations with stochastic behavior (from randomized algorithms) or unpredictable behavior (from complexity). We explicitly assign beliefs to logical statements, costs to computations, and rewards to problem outputs. Then, we analyze the problem to find an execution strategy that optimizes an expected utility function.

This approach improves the speed and quality of planning on multiple levels, from high-level reasoning (like how to balance expansion and refinement in multi-modal

planning) to low-level subroutines (like configuration testing). It also helps formalize and refine some performance enhancing techniques previously used in motion planning, such as lazy evaluation [13, 19, 113], most-constrained-first heuristics [59, 62, 75], and "fuzzy" scheduling [56, 101].

Driven by the empirical success of PRM motion planners, motion planning as a field has become more about "planner engineering" than "motion science". The literature offers many examples of "tweak-and-experiment" approaches: a certain modification to a randomized method demonstrates empirically superior performance in a handful of experiments. The performance improvements are argued to generalize to similar problems. But these conclusions are based on an appeal to intuition rather than explicit reasoning. Decision-theoretic planning changes more of this reasoning from implicit to explicit. Reasoning with explicit hypotheses helps merge planner engineering with motion science, by making planning faster while furthering our understanding of motion spaces.

1.5 Summary of Contributions

As contributions to this work, I:

- Describe and prove the completeness properties of the new Multi-Modal-PRM algorithm (Section 2.3).
- Develop the Incremental-MMPRM algorithm, an incremental variant of Multi-Modal-PRM with the same completeness properties, but is faster by orders of magnitude (Section 2.4).
- Implement the legged locomotion planner of Chapter 3, which extends work by Tim Bretl on a planar rock-climbing robot [21] to general robots in 3D [57, 58].
- Develop a technique that biases a locomotion planner with motion primitives, making motion more natural [55] (Section 4.1).
- Develop a manipulation planner for a humanoid robot [59], which contributes an

adaptive mode discretization (Section 2.2.2) and a utility-based motion strategy (Section 4.2).

- Optimize several planning subroutines using a principled decision theoretic approach (Chapter 5). Chapter 5 covers unpublished contributions as well as contributions presented in parts of published work (e.g. learning step feasibility in [56] and bottleneck-based subgoal selection in [59]).

Chapter 2

Multi-Modal Planning

Robotics systems with contact, such as legged locomotion and manipulation, move between *modes* where contact is fixed. In each mode, the allowable motions lie on a lower dimensional submanifold of configuration space. When an existing contact is broken (or a new contact is reached), the system switches to a new submanifold of different dimensionality. Take manipulation for example. If the manipulator is not touching the object, only the manipulator configuration parameters vary. The motion lies in a subspace of the configuration space of the manipulator/object system. Once the manipulator touches the object, the object moves with the manipulator. Both the manipulator and the object parameters vary, and the motion now lies in a different submanifold.

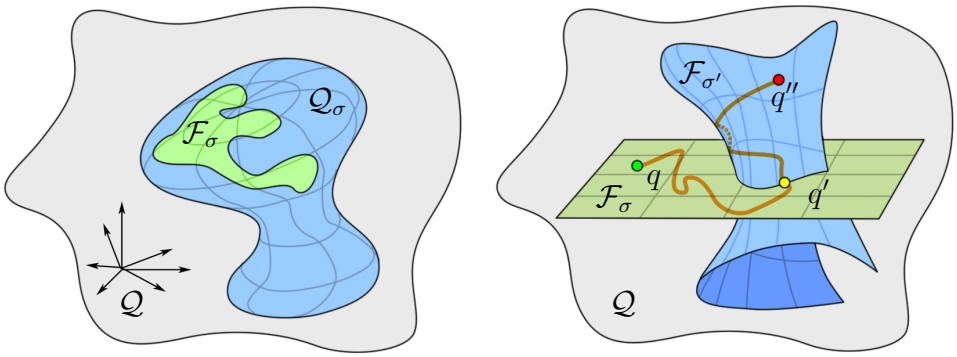
To plan for these and similar systems, we must find both a *discrete* sequence of mode switches and *continuous* single-mode motions to achieve them. This process is called *multi-modal planning*. Multi-modal planning may be difficult for several reasons. There may be a large or even uncountably infinite number of mode choices. Single-mode planning may be hard as well, because the feasible set of a mode may be nonlinear, high-dimensional, and have complex geometry. State-of-the-art techniques for planning in such spaces (e.g., probabilistic roadmaps [71]) can produce a feasible motion quickly when one exists, but cannot report in finite time that no motion exists. This is particularly troublesome in multi-modal planning, where a large number (thousands or more) of single-mode planning queries are infeasible.

Multi-modal planners have been implemented for specific problems, but techniques have not been compared across problems. In this chapter I study multi-modal planning in a general setting. I present two sample-based multi-modal planning algorithms, and a new completeness proof. The first algorithm, Multi-Modal-PRM, constructs roadmaps in all modes using random sampling. Given assumptions often satisfied in practice, it is provably *probabilistically complete*, which means that as more time is spent, the probability that it fails to finds a path asymptotically approaches 0. In fact, the bound decreases exponentially in the time spent planning, so it is in some sense fast.

However, Multi-Modal-PRM is a brute force approach, and is not practical when the number of modes is large. For most multi-modal planning problems, the number of mode switches needed to reach the goal is orders of magnitude smaller than the total number of modes that are considered by Multi-Modal-PRM. The second algorithm, Incremental-MMPRM, restricts exploration to a subset of modes, which grows incrementally according to some ordering. It has the same asymptotic performance as Multi-Modal-PRM, but it is much faster on most problems. I also present a method to produce a good mode ordering, by prioritizing modes that lead to the goal and are likely to contain feasible single-mode motions.

2.1 Background

A system configuration q lies in a configuration space $\mathcal{Q}$ of dimensionality $\dim(\mathcal{Q})$. Not all points in $\mathcal{Q}$ need to obey physical or operational constraints. For example, in legged locomotion, q may encode the joint angles of the robot and the translation and orientation of its body. Then, most configurations in $\mathcal{Q}$ place the robot high in the air or penetrating the terrain. A multi-modal planning problem asks for a path in $\mathcal{Q}$ that achieves a goal while obeying mode-specific constraints.



(a) (b)

Figure 2.1: (a) At a mode σ, motion is constrained to a subset $\mathcal{F}_\sigma$ of a submanifold $\mathcal{Q}_\sigma$ with lower dimension than $\mathcal{Q}$. (b) To move from configuration q at stance σ to q'' at an adjacent stance σ', the motion must pass through a transition configuration q'.

2.1.1 Modes

The system moves between a set of discrete modes, Σ. Each mode $\sigma \in \Sigma$ defines a *feasible space* $\mathcal{F}_\sigma$, the set of configurations that satisfy certain mode-specific constraints. These constraints can be divided into two classes.

- *Dimensionality-reducing* constraints, often represented as functional equalities $C_\sigma(q) = 0$. Define the submanifold $\mathcal{Q}_\sigma$ as the set of configurations that satisfy these constraints.

- *Volume-reducing* constraints, often represented as inequalities $D_\sigma(q) > 0$. These may cause $\mathcal{F}_\sigma$ to be empty. Otherwise, $\mathcal{F}_\sigma$ has the same dimension as $\mathcal{Q}_\sigma$ but lower volume (Figure 2.1a).

For example, in legged locomotion, σ is a fixed set of footfalls. Enforcing contact at the footfalls introduces closed-loop kinematic constraints, defining $\mathcal{Q}_\sigma$. Collision avoidance and stability constraints are volume-reducing, and restrict $\mathcal{F}_\sigma$ to a subset of $\mathcal{Q}_\sigma$.

2.1.2 Transitions

Suppose the system is at configuration q at mode σ. If the system is allowed to transition to mode σ', then we say σ and σ' are *adjacent*. "Allowed" simply means that a transition is not forbidden, not necessarily that a feasible motion connects σ and σ'. For example, in legged locomotion, σ' may have all the same footfalls as σ, minus one, and the transition from σ to σ' corresponds to breaking a contact. Computationally, adjacency testing is almost trivial.

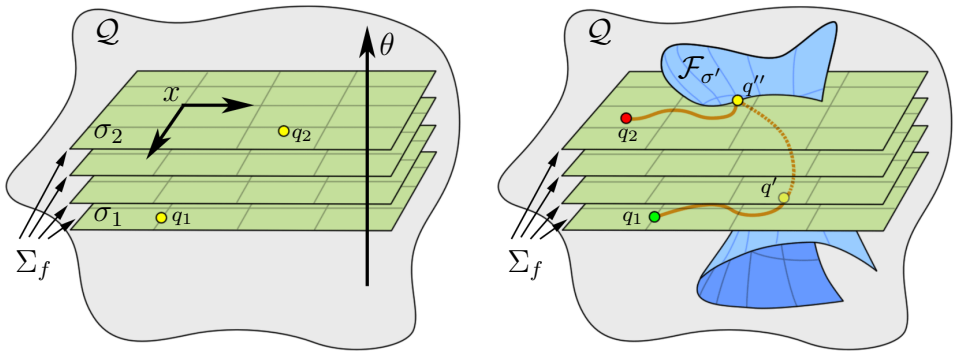
To switch to σ, the planner must plan a path in $\mathcal{F}_\sigma$ that ends in a configuration q' in $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ (Figure 2.1b). The region $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ is called the *transition* between σ and σ', and q' is called a *transition configuration*.

The existence of a transition configuration q' is a necessary, but not sufficient condition for a single-mode path to connect q and σ'. It may be the case that q and q' lie in different connected components of $\mathcal{F}_\sigma$. It is typically faster to sample a transition configuration than plan a single-mode path. This observation will be instrumental in selecting a good mode ordering for Incremental-MMPRM. Specifically, this observation suggests a "lazy" strategy that avoids single-mode planning until a transition configuration has been found.

2.1.3 Mode Families

For systems where Σ is infinite and uncountable, we partition Σ into F disjoint *families*, $\Sigma_1 \dots \Sigma_F$. Every mode $\sigma \in \Sigma_f$ is defined uniquely by a *coparameter* θ in a manifold Θ_f (Figure 2.2a). We say $\dim(\Theta_f)$ is the *codimension* of σ. To illustrate, consider manipulation of a single object using grasping. In one mode family, the manipulator does not touch the object, and in the other, the object is grasped. The coparameters of the first family describe the fixed location of the object, because each possible location of the object defines a mode. The coparameters of the second describe the location of the grasp on the object, and each distinct grasp location defines a mode.

Furthermore, no two modes of a family overlap. This means to switch between two modes of the same family, the system much switch modes to another family



(a) (b)

Figure 2.2: (a) A continuous family of modes Σ_f. Each value of the co-parameter θ defines a different mode $\sigma \in \Sigma_f$ (four are shown). Each mode is depicted as a linear subspace parameterized by x (but, in general, modes may be nonlinear). A mode in Σ_f can be identified by a representative configuration, e.g. q_1 identifies σ_1 and q_2 identifies σ_2. (b) Moving within Σ_f, from q_1 to q_2, requires transitioning to a different mode σ' at q', and back again at q''.

(Figure 2.2b). Also, any configuration q in $\sigma \in \Sigma_f$ identifies a single coparameter θ (Figure 2.2a). So, a mode $\sigma = (f, q)$ can be represented by an integer f to describe the family, and a *representative* configuration q from which the coparameter is uniquely derived.

This model is sufficiently general for most hybrid systems. For example, consider a system of N objects that can translate one-by-one on the infinite line, so $\mathcal{Q} = \mathbb{R}^N$. Let the coordinates of the fth object be x_f. Then a mode σ in the fth family allows object f to be moved while keeping all other objects fixed, and hence x_f varies. We describe σ using coparameters $x_1, \ldots, x_{f-1}, x_{f+1}, \ldots, x_N$, the positions of all objects besides f. Thus, each mode has codimension $N - 1$.

In legged locomotion, suppose the terrain surface is parameterized by two parameters (u, v), and all footfalls are point contacts. A robot with N feet has 2^N mode families, described as follows. A mode σ, with $n \leq N$ feet in contact with the terrain, assigns foot i_1 to touch the terrain at (u_{i_1}, v_{i_1}), foot i_2 to touch the terrain at (u_{i_2}, v_{i_2}), ..., and foot i_n to touch the terrain at (u_{i_n}, v_{i_n}). The mode belongs to a family Σ_f, corresponding to the foot indices $S_f = \{i_1, \ldots, i_n\}$. Σ_f has $2n$ coparameters

describing the positions of the feet, $\theta = (u_{i_1}, v_{i_1}, \ldots, u_{i_n}, v_{i_n})$.

Families usually arise when the dimension-reducing equality $C_\sigma(q) = 0$ can be written in the form $C_f(q) = \theta$. The submanifolds $\mathcal{Q}_\sigma$ are then the level sets of $C_f(q)$. In a rough mathematical sense, a family is a foliation, and its modes are the leaves.

2.1.4 Planning in a Single Mode

Probabilistic roadmap (PRM) planners and other sample-based methods (see Appendix or Chapter 7 of [37]) can be used to plan in a single mode. Applying PRMs to a submanifold $\mathcal{Q}_\sigma$ typically requires adapting three subroutines: distance computations, configuration sampling, and path segment feasibility testing. With these subroutines in place, PRMs usually converge quickly, as they do in Euclidean spaces.

Often, $\mathcal{Q}_\sigma$ can be directly parameterized using an atlas of charts [18, 39]). In the case where $\mathcal{Q}_\sigma$ is a linear submanifold, it can be treated as a Cartesian space with one chart, and these subroutines are trivial. If it is nonlinear, parameterization may be harder, requiring multiple charts (e.g., a robot subject to closed-chain kinematic constraints on multiple legs may require dozens of charts). Sampling $\mathcal{Q}_\sigma$ is straightforward, but computing distances and testing path segments require reasoning across charts. An alternative to parameterization is to embed the manifold in an ambient space, and move configurations onto the manifold using numerical methods [88]. This latter approach is taken in the locomotion planner of Section 3.3.4.

2.1.5 Planning in Multiple Modes

Most systems (but not all, e.g. [22]) are posed as having a continuous (uncountably infinite) set of modes. Section 2.2.2 describes various ways they can be discretized for planning. Given a discretized set of modes $\{\sigma_1, \ldots, \sigma_n\}$, the planner explores a *mode graph*. Here, each vertex represents a mode, and an edge exists between two adjacent modes. The planner may construct this graph partially and incrementally (for example, using search) if it is too large to represent.

To illustrate, one basic method incrementally builds a tree $\mathcal{T}$ of configurations reachable from the start. $\mathcal{T}$ is initialized to the start configuration. Each expansion

step picks a configuration q at mode σ in $\mathcal{T}$, enumerates each adjacent mode σ', then attempts to plan a feasible single-mode path from q to a transition configuration q' in $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. If successful, q' is added to $\mathcal{T}$. This is repeated until the goal is reached, and the single-mode paths leading up to the goal are concatenated together into a multi-modal path. This method is sometimes impractical due to a high branching factor. In general, it is unreliable because each single-step planning query may fail (see the discussion in Section 2.2.3) and it only considers a single transition configuration in each $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$, even though $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ may consist of several connected components.

2.2 Multi-Modal Planning Survey

In control theory, the systems considered in this thesis are known as *hybrid systems* because they mix continuous and discrete behavior. Hybrid system modeling, verification, controllability, and optimality have been studied heavily [17, 26, 27, 51, 99]. From a planning perspective, motion planners have been developed for part manipulation using multiple grasp and regrasp operations [2, 43, 76, 112], dexterous manipulation [35, 125, 128], legged locomotion [14, 22, 36, 57, 58, 81, 107], navigation among movable obstacles [102, 118], assembly planning [123], and reconfigurable robots [32, 33]. These planners have used a variety of approaches:

- *Path warping.* A constraint-violating path through $\mathcal{Q}$ is warped into $\mathcal{F}$ [43].
- *Macro actions.* The planner is restricted to use a set of mode-crossing "macro actions", reducing the system to a fully discrete or fully continuous one [36, 81, 107].
- *Switching actions.* This approach uses a sample-based planner, where a mode-switch can be executed at any sampled transition configuration (pp. 270–274 of [87], [125]).
- *Decoupled planning.* In a first stage, a sequence of modes is produced, and in the second, single-mode paths are planned [32, 33]. A common approach

in manipulation is to plan an object trajectory first, then motions to make or break contact with the object [35, 76, 128].

- *Controllability transformation.* Using controllability properties of certain systems, an invalid, continuously transitioning path is deformed into a valid, discrete set of single-mode paths [2, 112].

- *Methodical search.* Mode transitions are methodically explored, by combining graph search with single-mode planning [2, 14, 22, 57, 58, 59, 102, 101, 123].

Early work considered modes with low-dimensional [2, 118, 123] or geometrically simple [14] feasible sets. In such cases, multi-modal planning queries can be answered exactly by computing the connected components of each mode and the transitions between modes. Warping methods have had limited application, and are essentially a local optimization technique that easily gets trapped in local minima. Macro action approaches are best suited for repetitive tasks, such as walking on flat ground or structured terrain like stairs.

The rest of this section considers the remaining four approaches, particularly when modes have high dimension. We identify three major structural differences between these planners: first, whether mode transitions are reached implicitly or explicitly; second, how continuous (uncountable) mode sets are discretized; and third, how computation is allocated among multiple modes.

2.2.1 Mode Transitioning

Consider planning a motion that moves from σ to σ'. The endgame region for single-mode planning is the transition $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. We distinguish between explicit and implicit approaches to reaching the endgame region.

- An implicit *generate-and-test* approach generates paths in $\mathcal{F}_\sigma$, and tests if any endpoints reach $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. If so, the mode is switched.

- An explicit *mode-before-motion* approach samples one or more configurations q from $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$, and plans single-mode paths to reach them.

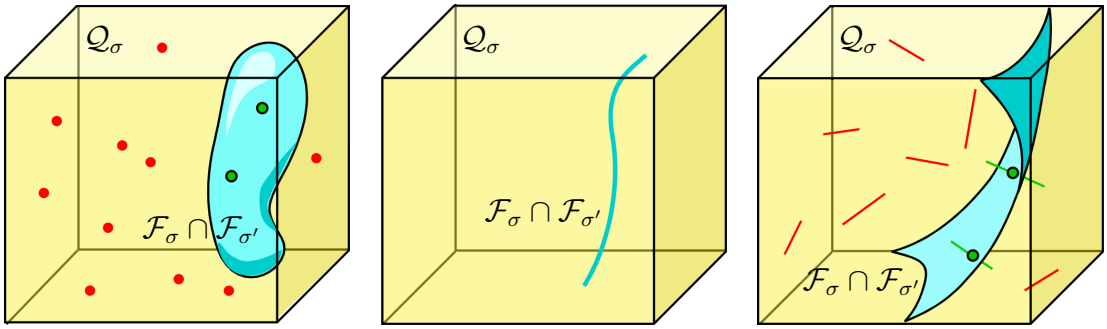


Figure 2.3: A single-mode planner in $\mathcal{Q}_\sigma$ (represented as a 3D cube) must terminate at the transition $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ (in blue). (a) If $\dim(\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}) = \dim(\mathcal{Q}_\sigma)$, arbutrary paths may terminate in the endgame, so implicit methods can be used. (b) If $\dim(\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}) < \dim(\mathcal{Q}_\sigma) - 1$, arbitrary paths have probability zero of reaching the endgame, so explicit methods must be used. (c) If $\dim(\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}) = \dim(\mathcal{Q}_\sigma) - 1$, sampled paths may intersect the endgame, so an implicit boundary-detection method can be used.

Switching action approaches use implicit transitioning. Decoupled planning, controllability transformation, and methodical search methods are usually explicit. Mixed strategies are also possible, where transitions are implicit in certain modes and explicit in others. For example, several planners for manipulation of objects that slide on a plane [35, 59, 128] use explicit methods for making contact, and implicit methods when breaking contact.

Depending on the dimension of $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$, implicit methods may not be viable. In the following discussion, assume a feasible path exists, and view the implicit method as sampling configurations (the endpoints of generated paths) from $\mathcal{F}_\sigma$, which terminates when a sample satisfies the endgame condition.

Endgame of same dimension. If $\dim(\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}) = \dim(\mathcal{F}_\sigma)$, then the endgame has nonzero volume relative to $\mathcal{F}_\sigma$, and the implicit method has a nonzero chance of sampling a transition configuration (Figure 2.3a). Similarly, the explicit method will eventually find a transition configuration if it samples from $\mathcal{Q}_\sigma$.

Endgame of lower dimension. If $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ is of lower dimension, it has zero volume relative to $\mathcal{F}_\sigma$. Thus, a configuration picked arbitrarily in $\mathcal{F}_\sigma$ has probability zero of meeting the endgame condition, and the implicit approach is not viable (Figure 2.3b). The explicit approach is still viable, but will need a specialized transition sampler that can sample $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ with nonzero probability.

Endgame of exactly one less dimension. In the case where $\dim(\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}) = \dim(\mathcal{Q}_\sigma) - 1$, another implicit approach could be used (Figure 2.3c). Here, the transition set draws a separating boundary in $\mathcal{Q}_\sigma$. An arbitrary path has nonzero probability of crossing the boundary, so a mode switch can be produced by detecting the exact point of crossing. This approach has only been used once in the surveyed literature [125].

Composite endgame over a mode family. Another possibility, when the modes are continuous, is to consider an endgame region $\mathcal{T}$ that consists of the union of *all* transitions to modes in a family. This increases the dimension of the endgame region by allowing the family's codimensions to vary. This may make implicit methods viable. For example, consider manipulation of objects that slide on a plane [35, 59, 93, 128]. Suppose the manipulator contacts the object at mode σ, and Σ_f contains all modes where the manipulator is not in contact. Since it is feasible to release contact at any time, $\mathcal{T}$ is exactly $\mathcal{F}_\sigma$, and the transition can be made implicitly at any point.

If both methods are viable, which one works faster depends on implementation details, and the shape and volume of $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. Results from Section 5.7 suggest that it may be faster to plan "out" of highly constrained regions (i.e., by explicitly starting the search from a transition) than to plan "into" them. So if the endgame is small (such as in dexterous manipulation or locomotion in rough terrain, where contacts must be placed precisely to be useful) explicit methods may be better. They may also help include mode-level reasoning (like search heuristics or feasibility estimates), because mode switches are chosen deliberately.

However, implicit methods may work better if a large portion of the endgame is not reachable from the start configuration, since explicit methods would lead to wasting time trying to connect unreachable configurations. They may also be faster for systems subject to nonholonomic constraints, because explicit methods must solve an expensive boundary value problem to reach a goal exactly.

2.2.2 Discretizing a Continuous Set of Modes

Discretization is typically implicit when mode switching is implicit. In decoupled planning, discretization is usually fixed to a small subset of modes [32, 33].

Controllability transform methods [2, 112] discretize by connected components of a certain set $\mathcal{T}$. Here, $\mathcal{T}$ is the set of *all* transitions between modes of two families. Paths through $\mathcal{T}$ are non-physical (making transitions infinitely often), but using a controllability condition, can be deformed into a sequence of single-mode paths. Then, any two modes in a connected component of $\mathcal{T}$ can be mutually reached. The algorithm then consists of two parts: identifying connected components of $\mathcal{T}$ (exactly [2] or approximately [112]), and then single-mode planning between connected components.

In methodical search, several approaches have been used:

- Prediscretization. Discretize before planning [2, 57, 101]. For example, in legged locomotion, the robot can be restricted to use a finite number of candidate footfalls sampled over the terrain.

- On-the-fly. The mode graph is built incrementally using search; a finite number of adjacent modes are sampled at each expansion step [58, 102]. For example, in legged locomotion, a set of candidate footfalls could be sampled for each step.

- Adaptive. The mode graph is grown by randomly sampling mode transitions [59], for instance, using strategies that try to distribute modes evenly across Σ.

The prediscretization and search-based methods introduce the dilemma of selecting a resolution. Too coarse, and the planner will not be able to find a solution; too fine, and the mode graph becomes extremely large. A connected component-based

discretization avoids this dilemma, but is not general. The adaptive discretization is general, and may produce a sparse sampling for easy problems, or a denser sampling for harder problems.

2.2.3 Allocating Single-Mode Computation

Even once modes are discretized, planning across modes remains challenging. There may be an enormous number of modes that need exploring, and no practical single-mode planner can determine if no feasible path exists between two configurations. As a PRM planner devotes more computation to a single mode, it is more likely to find a single-mode path. However, if no path exists, all this time is wasted. Various strategies have been used to allocate planning among modes.

First, allocation can be either sequential or interleaved.

- Sequential. Single-mode planning is performed for a certain number of iterations. If no path is found, the planner gives up and explores alternative modes [22, 32, 35, 59, 76].

- Interleaved. Single-mode planning is interleaved among multiple modes simultaneously [56, 58, 87, 101, 102, 125].

When many modes are infeasible, choosing a cutoff time for a sequential allocation is difficult. Too much, and multi-modal planning spends most of its time on infeasible modes. Too little, and the planner may miss important paths. Interleaved allocations help avoid a difficult tuning process.

Second, heuristics can be used to focus computation in promising modes.

- Distance-to-random-configuration heuristic. Modes are picked by sampling a random configuration and picking the mode that contains the closest milestone, as measured by a metric that incorporates inter-mode distances [87, 125]. This helps distribute modes evenly throughout configuration space.

- Mode-counting search heuristics [58, 102, 101, 118]. The planner's exploration of individual modes is prioritized by the estimated number of mode changes to

reach the goal. This helps the planner quickly find a sequence of modes that contains a path to the goal.

- Lazy strategies, which delay single-mode planning as long as possible [22, 58]. These speed up the planner overall by delaying expensive computations until they are absolutely needed.

- Importance sampling [55, 56, 59]. Single-mode planning is distributed nonuniformly, to focus more computation in modes that are likely to be feasible. This helps explore a large number of feasible modes quickly.

2.2.4 Discussion

A major motivation for this survey is to identify structural choices that are widely applicable across problems. For a basic implementation of a general-purpose multi-modal planner, I make the following recommendations:

- Make mode transitions explicitly. This is general-purpose and has additional advantages.

- Discretize modes adaptively. This is general-purpose and avoids a difficult choice of resolution.

- Allocate single-mode planning in parallel. This avoids having to tune PRM cutoff time, and will be shown below to have theoretical completeness properties.

2.3 Multi-Modal-PRM

This section presents a planner, Multi-Modal-PRM, that builds probabilistic roadmaps across all modes. It assumes modes have been discretized in advance. It is rather brute-force and impractical to use for a large number of modes, but is simple to analyze. This section summarizes its probabilistic completeness results, which are later proven in Section 2.5.

2.3.1 Algorithm

MULTI-MODAL-PRM builds PRMs across all modes, connecting them at explicitly sampled transition configurations (Figure 2.4). Suppose there are m modes, $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$. For each $\sigma_i \in \Sigma$ maintain a roadmap $\mathcal{R}_{\sigma_i}$ of $\mathcal{F}_{\sigma_i}$ (as in the basic PRM algorithm given in the Appendix). Define the sampler SAMPLE-MODE(σ_i) as follows. Uniformly sample a configuration q from $\mathcal{Q}_{\sigma_i}$. If q is in $\mathcal{F}_{\sigma_i}$, q is returned; if not, SAMPLE-MODE returns failure. Similarly, define SAMPLE-TRANS(σ_i, σ_j) to rejection sample from $\mathcal{F}_{\sigma_i} \cap \mathcal{F}_{\sigma_j}$. MULTI-MODAL-PRM is defined as follows:

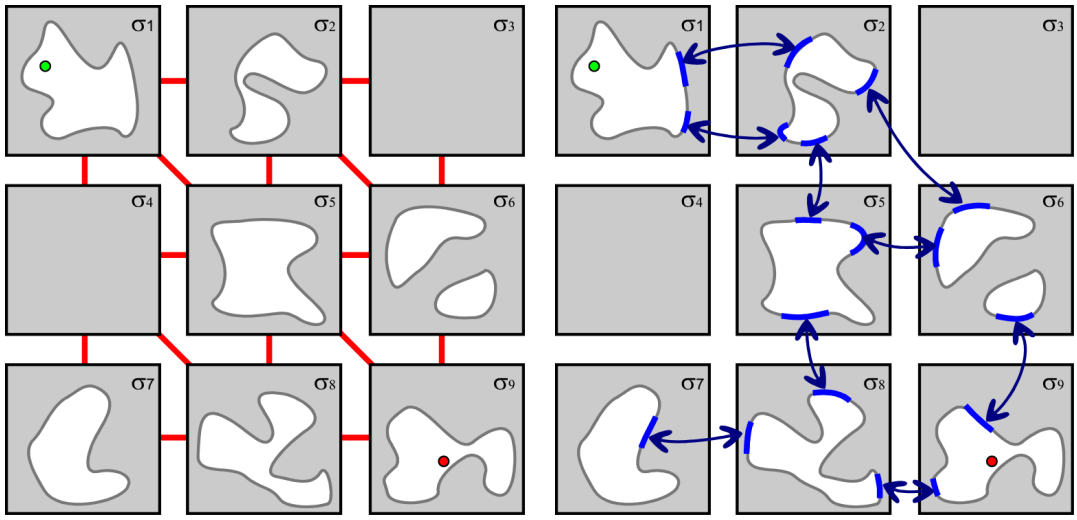
MULTI-MODAL-PRM(q_{start}, q_{goal}, N)
Add q_{start} and q_{goal} as milestones to the roadmaps corresponding to their modes (σ_{start} and σ_{goal}).
Repeat N times:

1. For each mode σ_i, sample a configuration q using SAMPLE-MODE(σ_i). If it succeeds, add q to $\mathcal{R}_{\sigma_i}$ as a new *milestone*, and connect it to existing milestones.

2. For each pair of adjacent modes σ_i and σ_j, sample a configuration q using SAMPLE-TRANS(σ_i, σ_j). If it succeeds, add q to $\mathcal{R}_{\sigma_i}$ and $\mathcal{R}_{\sigma_j}$, and connect it to all visible milestones in $\mathcal{R}_{\sigma_i}$ and $\mathcal{R}_{\sigma_j}$.

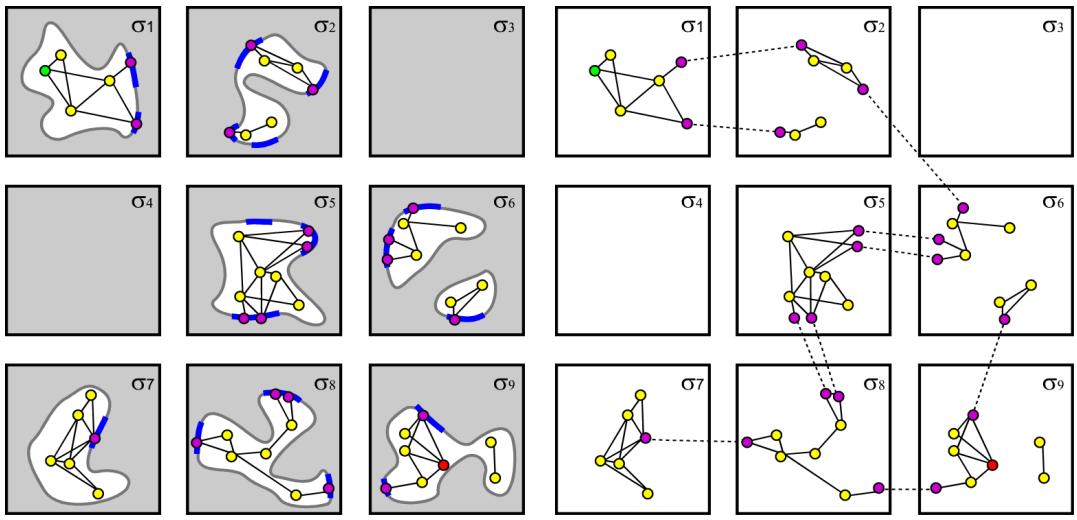
Build an aggregate roadmap $\mathcal{R}$ by connecting roadmaps at matching transition configurations.
If q_{start} and q_{goal} are connected by a path in $\mathcal{R}$, terminate with success. Otherwise, return failure.

Figure 2.4 depicts the operation of MULTI-MODAL-PRM in a fictitious nine mode example. Figure 2.4a illustrates that mode feasible spaces may be empty ($\mathcal{F}_{\sigma_3}$ and $\mathcal{F}_{\sigma_4}$) or disconnected ($\mathcal{F}_{\sigma_6}$). Figure 2.4b shows that transitions may be empty ($\mathcal{F}_{\sigma_1} \cap \mathcal{F}_{\sigma_5}$, for example) or disconnected ($\mathcal{F}_{\sigma_1} \cap \mathcal{F}_{\sigma_2}$). After individual roadmaps are built (Figure 2.4c), the aggregate roadmap is formed (Figure 2.4d). Even though no



(a) An abstract example problem with nine modes. The feasible spaces are white. Adjacencies are shown in red. The start and goal configurations are shown in green and red, respectively.

(b) Transition regions are highlighted. Arrows indicate how they map between modes.



(c) Building roadmaps. Yellow dots are milestones sampled from modes, purple ones are sampled from transitions.

(d) The aggregate roadmap. Transitions connected by dashed lines are identified.

Figure 2.4:

transition configuration was sampled between σ_2 and σ_5, a path was found connecting the start and the goal through σ_6.

2.3.2 Theoretical Properties

Section 2.5 proves that, under certain conditions, the probability that MULTI-MODAL-PRM gives an incorrect anwer (returns failure when a feasible path actually exists) is less than $c\exp(-dN)$, where c and d are positive constants and N is the number of iterations[1]. This means that as more time is spent planning, the probability of failure decreases quickly to zero. The constants c and d do not explicitly depend on the dimensionality of the configuration space, or the total number of modes. Furthermore, since a constant number of samples ($m+n$, where n is the number of adjacencies) are drawn per iteration, MULTI-MODAL-PRM also converges exponentially in the total number of samples drawn.

The convergence conditions are as follows:

1. The set of modes $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ is finite.
2. If $\mathcal{F}_{\sigma_i}$ is nonempty, then it is *expansive* (see below).
3. If $\mathcal{F}_{\sigma_i}$ is nonempty, SAMPLE-CONFIG(σ_i) succeeds with non-zero probability.
4. If $\mathcal{F}_{\sigma_i} \cap \mathcal{F}_{\sigma_j}$ is nonempty, SAMPLE-TRANSITION(σ_i, σ_j) samples each connected component of $\mathcal{F}_{\sigma_i} \cap \mathcal{F}_{\sigma_j}$ with non-zero probability.

The *expansiveness* property in condition 2 was introduced in [64] to characterize the convergence rate of PRM planners. Specifically, if a space $\mathcal{F}$ is *expansive*, then the probability that a PRM fails to connect two configurations decreases exponentially in the number of milestones. See Appendix for a formal definition of expansiveness and a statement of the convergence proof.

It is not guaranteed that any $\mathcal{F}_{\sigma_i}$ will be expansive, although a perturbation argument shows that non-expansiveness spaces are unlikely to exist in practice [34].

[1]This could also be written in the form $\exp(-d(N-K))$, where $K = \frac{\log c}{d}$. This form shows that for up to some minimum number of iterations K, MULTI-MODAL-PRM may have probability of failure up to 1.

If a space contains "narrow passages", it may have a low expansive measure, and PRM planning may be slow. In non-expansive spaces, the bound on the PRM failure probability takes the meaningless value of 1. If a space only contains cusps, it is non-expansive, but PRMs might still work well everywhere away from the cusps, since removing a tiny regions around the cusps makes the space expansive without changing its connectivity. But if the space contains regions of varying dimensionality, PRM planners have probability zero of answering most queries. In fact, this type of non-expansive space *is itself multi-modal*.

2.3.3 Discretization and Completeness

Probabilistic completeness holds if a solution path exists in the set of discretized modes. If no such path exists, the problem may have been discretized poorly (e.g., if a legged locomotion planner was restricted to an insufficiently high number of footfalls) or genuinely infeasible. Proving overall completeness of planners for continuous-mode systems is a matter for future work. By way of analogy from theoretical results in uni-modal planning, I pose the following conjectures:

- Fixed discretizations can be used in a probabilistically complete planner, by increasing resolution until a solution is found [8]. Such a planner will be probabilistically complete, but with rate decreasing exponentially in mode codimension.

- Adaptive discretizations can be used in a probabilistically complete planner. The rate of convergence will be high for certain "well-behaved" systems, and will have no explicit dependence on dimensionality. Theoretical results in [67] may be helpful in proving this.

2.4 Incremental Planner

Even executing a few iterations of Multi-Modal-PRM is impractical if the number of modes is large. Typical legged locomotion queries have over a billion modes,

but only tens or hundreds of steps are needed to go anywhere within the range of visual sensing. The INCREMENTAL-MMPRM variant uses heuristics to produce a small subset of modes that are likely to contain a path to the goal. Limiting planning to these modes make planning much faster. But heuristics are not always right, so INCREMENTAL-MMPRM is designed to gracefully degrade back to MULTI-MODAL-PRM if necessary.

2.4.1 Algorithm

INCREMENTAL-MMPRM alternates between *refinement* and *expansion*. At each round r, it restricts itself to building roadmaps over a candidate set of modes Σ_r. We set Σ_0 to the empty set, and all single-mode roadmaps (except the start and goal) are initially empty. It repeats the following for rounds $r = 1, \ldots, R$:

1. *Expansion*. Add new modes to Σ_{r-1} to produce the next candidate mode set Σ_r.

2. *Refinement*. Incrementally build roadmaps in Σ_r, by performing n_r mode and transition samples and adding them to the appropriate roadmaps (as in MULTI-MODAL-PRM).

The performance of INCREMENTAL-MMPRM depends mainly on the expansion heuristic. The heuristic does not affect asymptotic convergence, as long as the candidate mode set grows until it cannot expand any further (at which point INCREMENTAL-MMPRM behaves exactly like MULTI-MODAL-PRM). But practically, it has a large impact on running time. The heuristic below can be applied to any multi-modal planning problem, and improves running time by orders of magnitude.

2.4.2 Expansion: Search Among Feasible Transitions

For some systems, any set of modes leading to the goal (such as those found with heuristic search) could be a reasonable choice of candidate modes. But in many systems, *most* modes and transitions are infeasible, so this approach would cause the planner to waste most of its time building roadmaps in infeasible modes. If

the expansion step produces candidate modes that are likely to contain a feasible path, overall planning speed will be improved, even if expansion incurs additional computational expense.

Search among feasible transitions (SAFT) uses the existence of a feasible transition as a good indication that a feasible path exists as well (as in [21]). SAFT incrementally builds a mode graph $\mathcal{G}$, but without expanding an edge until it samples a feasible transition configuration. To avoid missing transitions with low volume, SAFT interleaves transition sampling between modes (as in [56]). It maintains a list $\mathcal{A}$ of "active" transitions. Each transition T in $\mathcal{A}$ has an associated priority $p(T)$, which decreased as more time is spent sampling T. The full algorithm is as follows:

SAFT-INIT (performed only once at round 0)
Add the start mode σ_{start} to $\mathcal{G}$. Initialize $\mathcal{A}$ to contain all transitions out of σ_{start}.

SAFT-EXPAND(r) (used on expansion round r)
Repeat the following:

1. Remove a transition T from $\mathcal{A}$ with maximum priority. Suppose T is a transition from σ to σ'. Try to sample a configuration in $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$.
2. On failure, reduce the priority $p(T)$ and reinsert T into $\mathcal{A}$.
3. On success, add σ' to the mode graph $\mathcal{G}$. For each transitions T' out of σ', add T' to $\mathcal{A}$ with initial priority $p(T')$.

Repeat until $\mathcal{G}$ contains a sequence of modes (not already existing in Σ_{r-1}) connecting σ_{start} to σ_{goal}. Add these modes to Σ_{r-1} to produce Σ_r.

As in [56], SAFT-EXPAND may find paths more quickly if the initial $p(T)$ estimates the probability that T is nonempty. Other heuristics, such as a bias toward easier steps, can also be incorporated into $p(T)$ [58].

2.4.3 Refinement: Strategies to Improve Connectivity

The running time of INCREMENTAL-MMPRM is also substantially affected by the refinement strategy. First, we need to tune the sample count parameter n_r: if n_r is too high, the planner might waste time on a candidate set of modes that contains no feasible path; too low, and the planner will expand the candidate mode set unnecessarily. We set n_r proportional to the number of modes, with some small tuning of the proportionality constant.

Second, given a fixed n_r, the planner should allocate single-mode planning computations to maximize its chance of finding a feasible path. If a mode already contains a highly connected roadmap, additional samples are unlikely to improve connectivity. We use a strategy that biases the planner's sampling on modes with poorly connected roadmaps. Furthermore, if the aggregate roadmap $\mathcal{R}$ already contains paths connecting two modes separated by σ, additional planning in σ is unlikely to improve connectivity. Thus, we bias sampling toward modes that could potentially connect large components of $\mathcal{R}$.

Also, rather than use the BASIC-PRM algorithm (see Appendix A) for single-mode planning, we use the SBL variant, which is much faster in practice [113]. SBL grows trees rooted from transition configurations, and delays checking the feasibility of straight line paths.

Results For large problems (millions of modes or more), SAFT explores far fewer modes (on the order of ten-thousands), and INCREMENTAL-MMPRM plans single-mode paths in even fewer (on the order of hundreds). This has been demonstrated in experiments on legged locomotion problems [18, 58] and will again be shown in the following chapter. Experiments also show that the connectivity-based refinement strategies are several times faster than the basic sequential sampling of MULTI-MODAL-PRM.

SAFT works well because for a modest additional expansion cost, it greatly reduces the amount of time spent in refinement. This overall strategy can be considered "lazy" because it delays single-mode planning, which is expensive compared to transition sampling. Furthermore, the existence of a feasible transition is a good indication

that a single-mode path exists (up to 90% in some problems [22]), and additional PRM planning provides little additional information about path existence or non-existence. In fact, if these assumptions generally hold, we can show that delaying single-mode planning is optimal (see Section 5.4).

2.5 Completeness Proofs

This section will prove two theoretical results. First, Multi-Modal-PRM is probabilistically complete and exponentially convergent in the number of iterations (and the number of samples as well). Second, Incremental-MMPRM is probabilistically complete even for a countably infinite number of modes, but subexponentially convergent in the number of samples.

The proof of the convergence of Multi-Modal-PRM has three parts. First I show that using rejection sampling, PRM planning is exponentially convergent in the number of samples drawn. Then I show that roadmaps connects transition components with exponential convergence. Finally I show that for any sequence of modes which contains a feasible multi-step path, the start and end modes are connected with exponential convergence.

2.5.1 Definitions

A process is *exponentially convergent* in N if the probability of failure is less than ce^{-bN} for some positive constants c and b. A useful composition principle allows us to reason about exponential convergence without stating the coefficients c and b, which become cumbersome. If two quantities are subject to exponentially decreasing upper bounds, their sums and products are themselves subject to new exponentially decreasing bounds. The product is trivial, and $c_1e^{-b_1N} + c_2e^{-b_2N}$ is upper bounded by ce^{-bN} for $c = (c_1 + c_2)$ and $b = \min\{b_1, b_2\}$. Therefore, if two processes are exponentially convergent in N, the probability that both of them succeed also converges exponentially in N. This can be extended to any composition (conjunctions and disjunctions) of a finite number of exponentially convergent processes.

2.5.2 PRM planning converges under rejection sampling

The first lemma proves that PRM planning converges exponentially, not just in the number of feasible milestones (as was proven in [64]), but also in the number of rejection samples drawn. Let $\mathcal{S}$ be a set of N configurations randomly sampled from $\mathcal{Q}$. Let p be the probability that a random sample from $\mathcal{Q}$ lies in $\mathcal{F}$. Let the milestones $\mathcal{M}$ be the set of feasible configurations from $\mathcal{S}$, and two feasible query configurations q_1 and q_2.

Lemma 2.5.1. *If $\mathcal{F}$ is expansive and $p > 0$, a roadmap $\mathcal{R}$ constructed from $\mathcal{M}$ connects q_1 and q_2 with probability exponentially convergent in N.*

Proof. Since the configuration samples are independent, the size of $\mathcal{M}$ is binomially distributed. Hoeffding's inequality gives an upper bound to the probability that $\mathcal{M}$ has n or fewer milestones:

$$Pr(|\mathcal{M}| \leq n) \leq \exp\left(-2\frac{(Np-n)^2}{N}\right).$$

By [64], if $\mathcal{F}$ is expansive, then the probability that a roadmap containing n uniformly sampled milestones fails to connect q_1 and q_2 is no more than $c\exp(-dn)$, for positive constants c and d. This bound also holds if the number of milestones is greater than n.

On the other hand, if $\mathcal{R}$ contains n or fewer milestones, the probability of failure is at most 1. Since these events are mutually exclusive, we have the overall probability of failure ν

$$\begin{aligned}\nu \leq & Pr(|\mathcal{M}| \leq n) \cdot 1 \\ & + Pr(|\mathcal{M}| > n) c\exp(-dn) \\ & \leq \exp(-Np^2/2) + c\exp(-Ndp/2) \\ & \leq (c+1)\exp(-N\min(p,d)p/2).\end{aligned} \tag{2.1}$$

where we have set $n = pN/2$ and used the composition principle. As desired, this bound is exponentially decreasing in N. □

2.5.3 Convergence of paths connecting transition components

Here we consider the probability of finding a path in $\mathcal{F}$ between arbitrary connected subsets A and B, by building a roadmap using N rejection samples in A, B, and $\mathcal{F}$.

Let $\mathcal{M}$ and p be defined as in Lemma 2.5.1. Suppose we rejection sample A and B respectively by drawing samples uniformly from supersets A' and B', with probability of success p_A and p_B. From N configurations sampled from A', let the milestones $\mathcal{M}_A$ be the configurations in A. Define $\mathcal{M}_B$ similarly.

Lemma 2.5.2. *Suppose $\mathcal{F}$ is expansive, and p, p_A, and p_B are nonzero. Let $\mathcal{R}$ be the roadmap constructed from all milestones $\mathcal{M}$, $\mathcal{M}_A$, and $\mathcal{M}_B$. If A and B are in the same connected component in $\mathcal{F}$, then the probability that $\mathcal{R}$ contains a path between A and B is exponentially convergent in N.*

Proof. View any pair of milestones q_A in $\mathcal{M}_A$ and q_B in $\mathcal{M}_B$ as PRM query configurations. Then $\mathcal{R}$ contains a path between A and B when (event X) $\mathcal{M}_A$ is nonempty, (event Y) $\mathcal{M}_B$ is nonempty, and (event Z) the roadmap formed from $\mathcal{M}$ connects q_A and q_B.

The probability that N rejection samples from A' fails to find a milestone in A is at most $(1-p_A)^N \leq e^{-Np_A}$. So event X is exponentially convergent. The same holds for Y by symmetry. Finally, event Z is exponentially convergent in N by Lemma 2.5.1 and since q_A and q_B are in the same connected component. The lemma holds by composition of X, Y, and Z.

□

2.5.4 Convergence of Multi-Modal-PRM

Theorem 2.5.3. *Let the assumptions at the beginning of Section 2.3 hold. Between any two configurations, if* any *feasible multi-step path exists, there is* some *feasible path that makes a finite number of mode switches.*

Proof. Expansiveness implies ϵ-goodness (see Appendix), which implies that each connected component of the feasible space has volume at least $\epsilon > 0$. Let ϵ_0 be such

that each mode is ϵ_0-good. Then, each mode can only contain $1/\epsilon_0$ components, with m/ϵ_0 components overall. □

Theorem 2.5.4. *Let the assumptions at the beginning of Section 2.3 hold. If two modes can be connected with a feasible multi-step path, then the probability that* Multi-Modal-PRM *connects the modes is exponentially convergent in the number of iterations N.*

Proof. If two modes can be connected with a feasible path, there is a feasible path with a finite number of mode switches. Let $y(t)$ be such a path.

Suppose the path travels through mode σ_k, starting at transition connected component T_1 and ending at T_2. Sample-Config and Sample-Transition have properties allowing Lemma 2.5.2 to be applied to roadmap $\mathcal{R}_{\sigma_k}$. Therefore, the probability that $\mathcal{R}_{\sigma_k}$ connects T_1 and T_2 exponentially converges to 1.

The theorem follows from repeating this argument for all modes along the path, and using the composition principle. □

Dimensionality and the total number of modes m do not explicitly affect the coefficients in the convergence bound (although the total cost per iteration is at least linear in m). The modes' expansiveness measures and the parameters p and p' have a straightforward effect on the convergence rate: when expansiveness or the parameters increase, the bound moves closer to zero. The bound also moves closer to zero if fewer modes are needed to reach the goal, or if the goal can be reached via multiple paths.

2.5.5 Convergence of Incremental-MMPRM

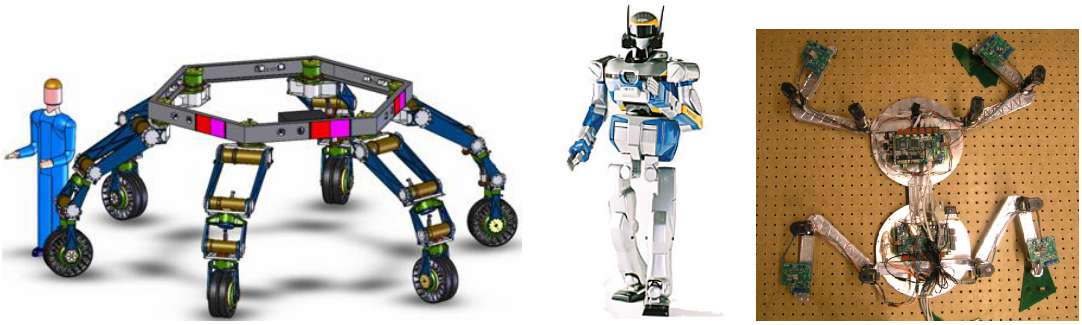
If the number of modes m is finite, then (by reduction to Multi-Modal-PRM) Incremental-MMPRM converges exponentially in the total number of samples N, as long the number of mode and transition samples is asymptotically proportional to N. If the number of modes is infinite, then Incremental-MMPRM may still converge as long as, in finite time, it considers a candidate subset of modes that contains a path to the goal. If the subset of modes continues to grow without bound, proving probabilistic completeness requires a more subtle treatment, which I leave open for future work.

Chapter 3

Legged Locomotion Planning

This chapter presents an instantiation of the INCREMENTAL-MMPRM algorithm of Section 2.4 in a legged locomotion planner. In legged locomotion, the motion of the robot is divided into discrete steps, where each step makes or breaks one contact. Each step occurs at a mode defined by a fixed stance, a fixed set of contacts with the environment. The planner uses a *mode-before-motion* approach, methodically searching the mode graph to select which contacts to make or break. This is well-suited for rocky, irregular, steep, or cluttered environments where the choice of contact matters greatly. This work was originally based on a planner for a planar rock-climbing robot [18]. Extending it to general 3D robots required new work in system modeling, testing motion constraints, faster configuration sampling routines, footstep selection heuristics, and stance pruning [57, 58]. It also integrates the probabilistically complete INCREMENTAL-MMPRM algorithm, which makes it more reliable.

The planner works with 3D robots of arbitrary morphology on arbitrary terrain. In particular, it has been applied to the ATHLETE six-legged robot, the HRP-2 humanoid robot, and the Capuchin free-climbing robot. It enables ATHLETE to use all six feet for maximum stability, or fewer feet for faster movement. Similarly, it allows HRP-2 to walk bipedally on easy terrain, or use hands or knees for support in difficult terrain. Experiments in simulation show that the planner can produce motions that traverse irregular, uneven, and steep terrain where gaits fail. It can be adapted easily to new robot morphologies. To illustrate this point, an example will be shown where



(a) (b) (c)

Figure 3.1: (a) Schematic of the ATHLETE robot, with a 1.8m human figure for size comparison. (b) The HRP-2 humanoid robot. (c) The Capuchin robot resting on four holds.

ATHLETE rappels down a rocky cliff face using a tether.

3.1 Robot Applications

The planner has been used on the following three robots.

Athlete ATHLETE (All-Terrain Hex-Limbed Extra-Terrestrial Explorer) is a large, 850kg, six-legged lunar vehicle developed by NASA's Jet Propulsion Laboratory (Figure 3.1a). Each of the six legs is nearly 2m long and has six revolute joints, ending in a wheel. On smooth terrain, ATHLETE may drive at high speed with the legs acting as suspension. On irregular or steep terrain ATHLETE's wheels may be braked so it can walk like a standard legged robot. Walking locomotion is more suited for the Moon's craters and mountains, which are likely to be targets for scientific exploration. Fixed gaits (e.g., an alternating tripod gait) have trouble traversing such terrain.

At first, ATHLETE will be teleoperated by human operators. However, ATHLETE has 36 joints and a non-human-like morphology, which makes direct control difficult and frustrating. Operators have taken hours to command the robot to walk short distances on even modest slopes. In Section 3.4, I will compare two ways of using motion planning to assist a human operator: first, by planning single-mode motions to reach

footfalls individually chosen by the operator, and second, by multi-modal planning to reach operator-chosen goal locations several body lengths away. In difficult terrain, the multi-modal planner is faster than a human.

HRP-2 The HRP-2 (Humanoid Robotics Platform) robot, developed by AIST Japan, is a 1.5m-tall, 58kg bipedal humanoid robot [70]. It has six joints in each of its arms and legs, and two joints in waist and neck (Figure 3.1b). The robot is a prototype to study autonomous or semi-autonomous operation in homes, offices, or construction sites. Thus, it should be able to plan its own motions in unstructured, cluttered environments.

Its humanoid form poses two additional challenges for planning: first, when walking bipedially, stability is delicate, so finding a stable configuration is harder; second, the planned motions should look natural to human operators. The first issue is addressed by the sampling algorithms of Section 3.3.3. The second will be addressed in the next chapter.

Capuchin The Capuchin robot (see Figure 3.1c), developed in the AI Lab at Stanford, is a 6kg, four-limbed free-climbing robot [129]. Each limb is 0.356m long, has two revolute joints, and has human-like joint limits. It is an experimental testbed for investigating sensing, planning, and control integration in steep environments. It climbs a steeply inclined plane, balancing itself by contacting the environment with "fingers" (small protrusions at the ends of its limbs).

The planner produces trajectories that are executed using a force-balancing controller [97]. In extremely steep terrain, useful contacts are even sparser than they are in rocky terrain. This actually makes mode-before-motion planning somewhat easier, because the planner is not overwhelmed by choice. But relatively few modes have non-empty feasible spaces.

3.1.1 Modeling Assumptions

The planner presented in this chapter makes several modeling assumptions:

- *Robot.* The robot links are rigid, with known geometry (such as from a CAD model). The mechanism is a tree-structured chain.
- *Environment.* The environment is rigid and has known geometry.
- *Coulomb friction.* A Coulomb friction model is assumed, with a known friction coefficient between the robot and environment. Here, the set of forces that can be applied at a point lie in a *friction cone.*
- *Discrete contacts.* Surface-to-surface contact is modeled using a finite set of point contacts. If the contact region is polygonal, planar, and has uniform friction coefficient, it can be represented exactly by point contacts at the vertices of the convex hull of the contact polygon. Contact regions that are curved or have nonuniform friction coefficient can be approximated to arbitrary precision by sampling contact points at the interface.
- *Quasistatic motion.* The robot is assumed to move quasistatically. This means it must stay quasistatically stable, where each configuration instantaneously satisfies equilibrium with gravity. Effectively, this means time and dynamic effects are ignored, and velocity is always assumed to be zero.
- *Fixed footfalls.* Contact are made and broken at discrete points in time; they may not slide or roll.

The assumption of known environment geometry and friction is likely the most restrictive. In most applications, the terrain will not be sensed accurately far away from the robot. This can be addressed by replanning when better models are available. More critically, small sensing errors could cause robots (especially bipedal ones) to fall, or collide with obstacles. Deformable or dynamic environments have a similar effect. This can be partially addressed in the planner by requiring the motion to satisfy more stringent stability and collision constraints, such as by shrinking the support polygon or requiring a minimum separation distance between the robot and the environment. A better solution would be to incorporate the planner with a reactive controller which can adapt to these errors.

Quasistatic motion is desirable for applications such as extraterrestrial exploration, where communication delays between the robot and human operators make dynamic motion risky. In postprocessing, a quasistatic path can be converted into a dynamically feasible trajectory one by finding a time parameterization that satisfies dynamic constraints [11]. But this technique is problematic for bipedal walking, because the robot must move extremely slowly to track a path while maintaining dynamic balance. Other techniques vary the trajectory as well as time, making motion more natural [69]. These methods cannot produce highly dynamic motions, such as running or jumping, which require anticipatory momentum buildup (such as flexing of the knees before a jump). These types of motions will not be addressed here.

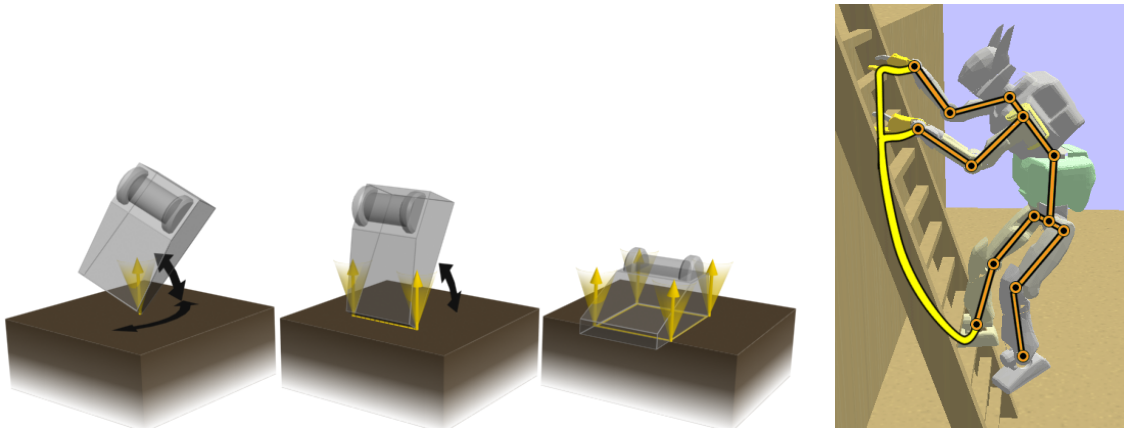
3.2 Motion Constraints

A legged robot's motion is largely dominated by the interplay of equilibrium and contact constraints. When many legs are in contact with the environment, there are more ways to distribute contact forces, and a wider range of configurations are stable. However, each new contact imposes additional closed-loop kinematic constraints, which restrict the robot's range of motion. The planner must trade-off between these competing constraints when choosing a sequence of footfalls.

A configuration q of the robot can be represented in a 6+N dimensional configuration space $\mathcal{Q}$, with 3 degrees of freedom each for the translation and rotation of a selected root link, and N degrees of freedom for the joints. For example, HRP-2 has 30 joints, hence a 36-D configuration space, and ATHLETE has 36 joints, hence a 42-D configuration space.

A set of simultaneous footfalls is also called a *stance*. Each stance defines a *mode*; in this chapter, the two terms are interchangeable. The robot moves by making *steps*, single-mode paths that make or break a single footfall. At a fixed stance σ with $n \geq 1$ footfalls, the set of feasible configurations is the *feasible space* $\mathcal{F}_\sigma$. A feasible configuration q must satisfy the following constraints:

- *Contact*. The linkage of q must reach all of the footfalls in σ. This is expressed as a set of loop-closure equations (see Section 3.2.1).



(a) (b)

Figure 3.2: (a) Point, edge, and face contact. (b) Three contacts form a kinematic linkage with two closed loops.

- *Equilibrium.* A necessary condition is that the center of mass must lie above a region called the *support polygon* [22]. Joint torques must exactly balance contact forces and gravity, without exceeding torque limits (see Section 3.2.2).

- *Collision.* The robot must satisfy joint angle limits, avoid self-collision, and collision with the environment (except at regions designated for contact). We use techniques based on bounding volume hierarchies to perform collision checking, as in [52, 114].

The *stance submanifold* $\mathcal{Q}_\sigma$ is the set of all configurations q that satisfy the contact constraint. It is a nonlinear submanifold, with lower dimension than $\mathcal{Q}$. Equilibrium and collision constraints reduce the volume of the feasible space $\mathcal{F}_\sigma$, but not its dimensionality.

3.2.1 Contact

A *footfall* refers to a fixed placement of a robot link against the environment. The link is usually a "foot" but does not have to be; it may be be a hand, knee, or any other part of the robot's body available for contact. A contact region can be a point, edge, or face contact (Figure 3.2a). A point contact does not constrain rotation, while an

edge contact constrains rotation to lie about an axis, and a face contact fixes rotation entirely. To simplify presentation, we will assume here that all footfalls have face contact.

The n footfalls in stance σ define a linkage of multiple closed-loop chains (Figure 3.2b). Configuration q must satisfy loop-closure equations. Let $\mathcal{Q}_\sigma \subset \mathcal{Q}$ be the set of all configurations q that satisfy these equations. Except at singular configurations, this set $\mathcal{Q}_\sigma$ is a sub-manifold of $\mathcal{Q}$ of dimensionality $42 - 6n$ for ATHLETE and $36 - 6n$ for HRP-2, which we call the *stance manifold* (Figure 2.1a). This manifold is empty if it is impossible for the robot to reach the footfalls in σ, for example if two contact points are farther apart than the maximum span of two legs.

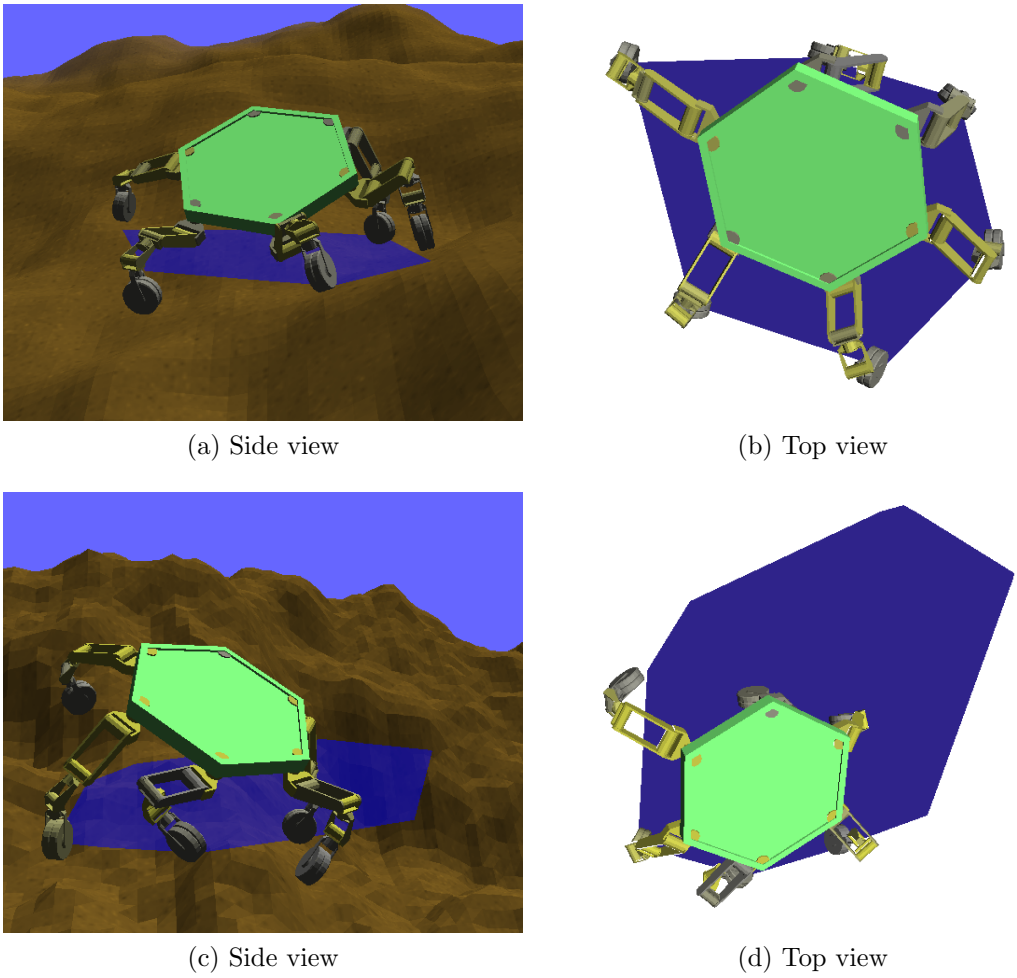
3.2.2 Equilibrium

As a first approximation, the robot can be considered a rigid object with all joints fixed. If there are no valid contact forces that keep the rigid object stable, then the robot cannot be either. Given the force of gravity, and a polyhedral approximation to the friction cone, the equilibrium condition for rigid objects can be tested efficiently using a linear program [22]. We can also efficiently compute the *support polygon*, a convex region in the horizontal plane over which the robot's center of mass must lie [20, 22] (Figure 3.3). These tests can quickly reject infeasible configurations of the robot.

A more refined stability test does not treat the robot as a rigid object, and considers the robot's bounded joint torques. The footfalls in σ define a set of contact points. The environment may apply forces at these points. Start from the generalized coordinate (Lagrangian) dynamics equations:

$$B(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + \sum_i J_i^T(q) f_i \tag{3.1}$$

where $B(q)$ is the positive definite mass matrix, $C(q, \dot{q})$ is the vector of Coriolis forces, $G(q)$ is the generalized gravity vector, and τ is the vector of joint torques. Here, f_i is the force applied by the environment at the i'th contact point, and $J_i(q)$ is the



(a) Side view

(b) Top view

(c) Side view

(d) Top view

Figure 3.3: Feasible configurations of ATHLETE on smooth and rough terrain. On easier terrain, the support polygon (shaded) corresponds to the convex hull of the projection of the contacts. On steeper, more irregular terrain, the support polygon may be very different.

Jacobian of the corresponding point on the robot. Since the robot is initially still, velocity and Coriolis forces are zero. Quasistatic equilibrium holds if

$$G(q) = \tau + \sum_i J_i^T(q) f_i \tag{3.2}$$

is satisfied (since $B(q)$ is positive definite, acceleration must be zero). The torques and contact forces must satisfy additional constraints,

$$\begin{aligned} &|\tau| \leq \tau_{max} \\ &f_i \in \mathcal{FC}_i \text{ for all } i \end{aligned} \tag{3.3}$$

where τ_{max} is the maximum applied torque and $\mathcal{FC}_i$ is the friction cone at the i'th contact point. Note that the components of τ_{max} corresponding to the translation and rotation of the root link will be zero. Using a polyhedral approximation to the friction cone, Eqs. (3.2,3.3) can be written as a linear program. Solving the LP produces a set of equilibrium forces $f_1, \ldots, f_n$ and torques τ. As a by-product, the joint torques τ can be fed to the robot's control system.

Other stability constraints (e.g., stability under force nondeterminism [104], external disturbances [103], or curved geometry [95]) could be easily included if desired.

3.3 Implementing a Multi-Modal Planner

The planner is an instantiation of the Incremental-MMPRM algorithm of Chapter 2. Implementing Incremental-MMPRM requires defining several subroutines specific to legged locomotion. First, it requires defining a stance and transition sampler. Second, it requires implementing a single-mode PRM planner, which requires configuration feasibility tests (already covered in Section 3.2), configuration sampling, and feasibility tests for path segments.

Transition-sampling search can be quite expensive due to the huge number of infeasible stances and transitions. Experiments show that it typically explores many tens of thousands of transitions, many of which are infeasible, even if only a few

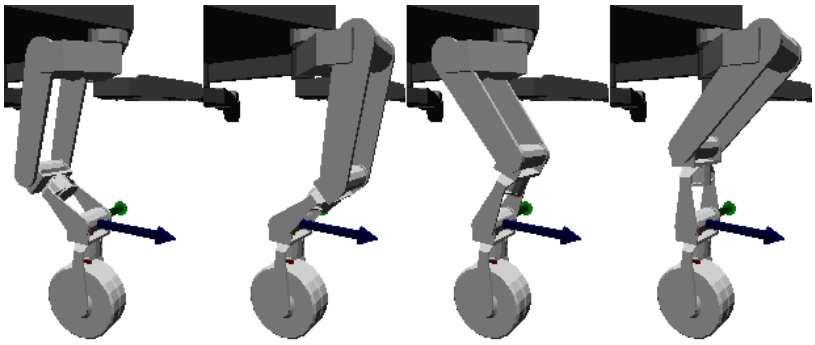


Figure 3.4: Each of ATHLETE's legs has eight IK solutions. Only four are shown here; the other four solutions appear identical, and are obtained by rotating the third and fifth joint angles are rotated 180° and reflecting the fourth joint angle about zero.

dozen steps are needed to reach the goal. To reduce the computation time wasted, we incorporate geometric reasoning to prune obviously infeasible transitions quickly (for example, transitions that add footfalls that are beyond the robot's reach). To direct the search to promising stances, we use several heuristics, including fuzzy planning, distance heuristics, and predictive learning. These subroutines are described below.

3.3.1 Stance Discretization

Even when the robot is restricted to use a small number of candidate footfalls, the resulting number of stances is large. For example, fixing 100 footfalls per leg for a six legged robot produces over 10^{12} stances. Thus, the planner does not explicitly represent all stances, but rather produces them on demand during search. Given a stance σ, it enumerates the adjacencies that remove a single footfall, and those that add a new footfall. Before planning, we define set of candidate footfalls `Footfalls` (typically a few hundred), and pick footfalls from `Footfalls` during planning. In our experiments, similar results have been observed when footfalls are sampled on-the-fly.

3.3.2 Stance manifold representation

For a stance σ, $\mathcal{Q}_\sigma$ can be sometimes directly parameterized using analytic inverse kinematics (IK). For example, if a foot is fixed by σ, then with ATHLETE's chassis

fixed, we can compute the joint angles of its leg to meet the specified foot placement. Including joint limits, each leg of ATHLETE has up to 8 IK solutions (see Figure 3.4), but could have none (for example, if the footfall is too far away from the chassis position). This parameterizes $\mathcal{Q}_\sigma$ with the transformation of the chassis and the joint angles of any free legs. If σ has n footfalls, this parameterization contains up to $8n$ charts.

Closed-form IK solutions only exist for specially designed geometry, and do not hold if the robot were to, say, make contact with a "knee" rather than a "foot". Thus, the planner typically avoids parameterizing $\mathcal{Q}_\sigma$.

3.3.3 Transition sampling

Consider the transition between σ and σ'. If σ' adds a new footfall, configurations in $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ may only use the footfalls in σ for support, while making contact with the new footfall in σ' and all footfalls in σ. Consequently, $\mathcal{Q}_{\sigma'} \subseteq \mathcal{Q}_\sigma$, and $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ has no greater dimension than $\mathcal{Q}_{\sigma'}$ and typically less volume than $\mathcal{F}_{\sigma'}$.

This suggests a basic two-step sampler. First, generate q in $\mathcal{Q}$ using an initial distribution ϕ. Then, solve an inverse kinematics (IK) problem so that the robot meets the footfalls in $\mathcal{Q}_{\sigma'}$. This latter step is necessary because a random sample from $\mathcal{Q}$ has zero probability of satisfying the contact constraints. However, experiments show that very few generated configurations are feasible. This is especially true when the support polygon of σ is small or the space around the robot is cluttered with obstacles. For example, when the HRP-2 robot walks on flat ground, our tests show that over 99.5% of samples generated on $\mathcal{Q}_{\sigma'}$ fail the equilibrium constraint. Thus, basic rejection sampling from $\mathcal{Q}_{\sigma'}$ averages several seconds to find a single feasible sample.

Iterative Constraint Enforcement

Instead, the planner uses an *iterative constraint enforcement* (ICE) technique that reduces the time to produce a feasible sample [57]. At a modest amount of additional computation, it greatly reduces the rejection rate. It represents the contact, support

NEWTON-RAPHSON-IK(q_0, σ)
1 $C(q) \leftarrow$ loop closure equations of σ
2 **for** $k = 0, \ldots, n$ **do**
3 **if** $||C(q_k)|| < \varepsilon$ **then**
4 **return** q_k
5 $d_k = -\nabla C(q_k)^\dagger C(q_k)$
6 Find $\alpha_k > 0$ such that $||C(q_k + \alpha_k d_k)|| < ||C(q_k)||$.
7 If no such α_k can be found, return "failure"
8 $q_{k+1} = q_k + \alpha_k d_k$
9 **return** "failure"

Figure 3.5: Newton-Raphson method to move a configuration q_0 onto $\mathcal{Q}_\sigma$.

polygon, and joint limit constraints of $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ as numerical equalities and inequalities, and uses an active-set, Jacobian-based solver. This simultaneously "repairs" violated constraints while solving IK.

Method ICE extends the Newton-Raphson method, a numerical method for solving nonlinear systems of equations which is widely applied in IK. Numerical IK techniques, in general, the loop closure condition as the solution set of a (vector valued) nonlinear equality $C(q) = 0$, and iteratively move a start configuration q_0 toward the solution set. In particular, the Newton-Raphson method (Figure 3.5) uses the pseudoinverse of the jacobian of C, denoted $\nabla C(\cdot)^\dagger$, to determine a local search direction.

We add inequalities that encode joint limit and equilibrium constraints. In particular, the joint limit and support polygon constraints can be written $D_{jl}(q) \geq 0$ and $D_{sp}(q) \geq 0$. To simultaneously solve these equations, we use an *active set* method [49]. At step k, $q = q_k$. Form an auxiliary equation

$$E_k(q) = \begin{bmatrix} C(q) \\ \hat{D}_{jp}(q) - \epsilon \\ \hat{D}_{sp}(q) - \epsilon \end{bmatrix} = 0 \tag{3.4}$$

where $\hat{D}_{jl}$ and $\hat{D}_{sp}$ only contain the inequalities of D_{jl} and D_{sp} that are violated at

Method	Analytic IK	Numerical IK	ICE
% Successfully solved	5.4	89	34.4
% Pass equilibrium	0.02	0.9	27.9
% Pass collision	0.02	0.4	26.0
Time / sample (ms)	0.83	9.1	69
Time / feasible sample (s)	4.2	2.3	0.27

Table 3.1: Transition sampling experiments for HRP-2 taking a step on flat ground

q_k. The value $\epsilon > 0$ is a separation distance that must be exceeded for a constraint to be removed from the active set. The method then takes a Newton-Raphson step,

$$q_{k+1} = q_k - \alpha_k \nabla E_k(q_k)^\dagger E_k(q_k)$$

where $\nabla E_k(\cdot)^\dagger$ is the pseudoinverse of the jacobian of E_k, and α_k is a step size chosen to reduce the residual of $E_k(q)$. This is repeated until convergence, or an iteration limit is reached.

Experimental Comparison Consider an example where the HRP-2 robot is in a transition between a two-legged and a one-legged stance on flat ground. Table 3.1 compares ICE to two methods that use IK alone: the Newton-Raphson method, and an analytic method. For all three methods, initial samples are generated using RLG [40], which uses simple reachability conditions to sample configurations close to satisfying loop-closure constraints.

Analytic IK finds solutions for about 5% of samples, while numerical IK succeeds on 89% of samples. This is because analytic IK only varies the six leg joints to meet the loop closure constraints, which requires that the body transformation be sampled such that both footfalls can be simultaneously reached. Numerical IK achieves a higher success rate by varying the body transformation as well. However, numerical IK is more computationally expensive per sample because it computes several matrix pseudoinverses. ICE is more expensive still, but the success rate is improved by an order of magnitude over numerical IK, and two orders of magnitude over analytic IK. This makes ICE faster per *feasible* sample than the other two methods. Similar

results are observed in experiments on ATHLETE.

Further speed gains are observed by first using analytic IK to find a configuration close to $\mathcal{Q}_\sigma$, which reduces the number of ICE iterations. Collision constraints could have also been included as inequalities in the ICE method to reduce the rejection rate further. However, it is difficult and expensive to compute penetration depths for non-convex geometry, making ICE slower in our experiments.

Initial Distributions

Sampling speed is further improved by using better initial distributions ϕ. The objective is produce configurations near $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$, which helps increase the success rate and reduce the number of ICE iterations. Consider the following two examples, which respectively use robot- and context-specific information.

Nominal configuration alignment ATHLETE is engineered to drive on rolling terrain with a *nominal configuration*, which can be perturbed significantly without violating constraints. In planning, we use this configuration to derive an initial distribution. First, we find a transformation T of the nominal configuration such that the links of footfalls in σ' are closely matched to their desired positions (by a least-squares fit [5]). Then we sample body transformations near T. Compared to RLG, this initial distribution makes ICE several times faster.

Seed sampling A feasible configuration at one stance suggests that a similar configuration may be feasible in a nearby stance. *Seed sampling* uses this idea to make sampling faster during the mode graph search. When considering the transition $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$, we examine transitions between σ and its adjacent stances to find an existing feasible configuration q. Distribution ϕ then samples q' in a neighborhood of q. In fact, experiments show a large fraction of feasible transitions can be found simply by directly executing ICE on the seeds (i.e. setting q' equal to q). In the plan of Figure 3.11, about 80% of the transitions are produced this way.

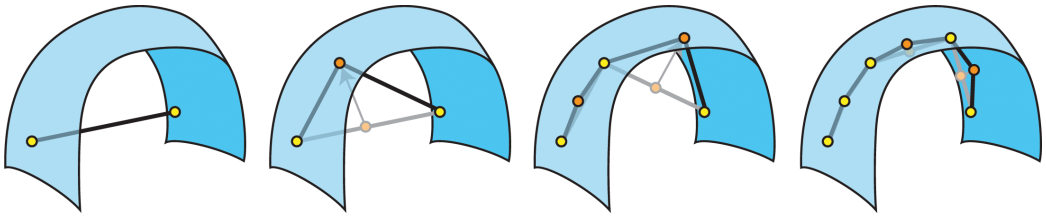


Figure 3.6: Deforming a straight line path onto the contact submanifold. The path is recursively bisected until a tolerance is reached.

3.3.4 Single-step motion planning

To connect two transition configurations q and q' in a stance $\mathcal{F}_\sigma$, we use a PRM variant called SBL [113], which grows two trees of milestones rooted from q and q'. As in the previous section, we face the difficulty of finding configurations on the manifold $\mathcal{Q}_\sigma$, which has zero volume in $\mathcal{Q}$. But this case is slightly different; SBL grows the trees by sampling in the neighborhood of an existing milestone q_0. Near q_0, $\mathcal{Q}_\sigma$ is approximated by the hyperplane

$$\{p \quad | \quad \nabla C(q_0)^T p = \nabla C(q_0)^T q_0\}$$

where $C(q) = 0$ is the set of loop-closure equations of σ. So we sample a configuration in a neighborhood on this hyperplane (as in [126]) before applying the Newton-Raphson method to move it onto $\mathcal{Q}_\sigma$.

To check straight-line paths between milestones q_1 and q_2 quickly, SBL uses a recursive bisection technique. The midpoint q_{mid} between q_1 and q_2 has the highest probability of being infeasible, so it should be tested first. However, a straight-line path will not, in general, lie in $\mathcal{Q}_\sigma$. Instead, we simultaneously deform the path onto $\mathcal{Q}_\sigma$ and check its feasibility, as follows (Figure 3.6). Apply the Newton-Raphson method to the midpoint of q_1 and q_2 to produce q_{mid} on $\mathcal{Q}_\sigma$. Then test if q_{mid} lies in $\mathcal{F}_\sigma$. If both steps succeed, recurse on both of the segments until a desired resolution has been reached; otherwise, the algorithm returns failure.

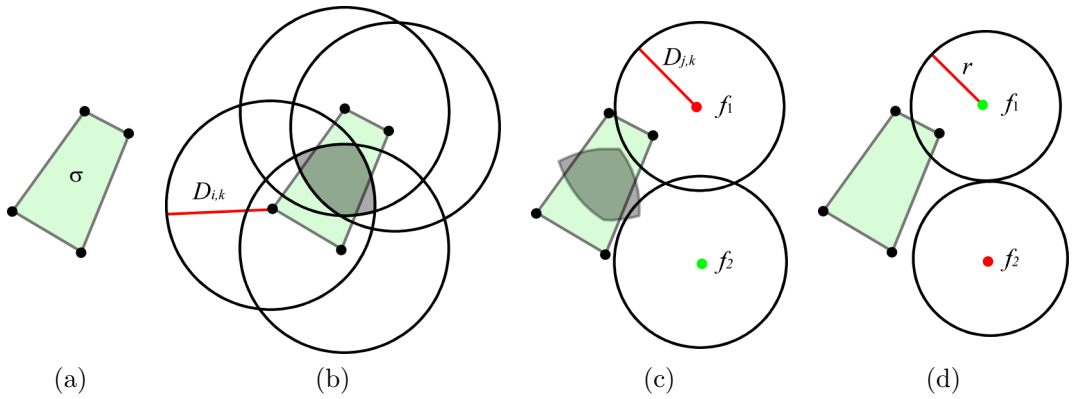


Figure 3.7: (a) Fixed points q_i for a stance σ. (b) Reach limits $D_{i,k}$ for a point p_k on link k. The region $\bigcap_i B(q_i, D_{i,k})$ is shaded. (c) A footfall on link j is reachable only if the fixed point q_j is within $D_{j,k}$ units of this region. This condition prunes f_1 but not f_2. (d) A footfall is reachable only if the fixed point is no more than distance r (given in the text) from the support polygon. This condition prunes f_2 but not f_1.

3.3.5 Pruning Infeasible Transitions

Recall that transition-sampling search explores the stance graph, expanding a stance σ to an adjacent stance σ' when a feasible transition configuration is sampled in $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. If the set of candidate footfalls `Footfalls` is large, then the number of stances adjacent to σ may also be large. However, most of these transitions are infeasible, and any time spent sampling them is wasted. For example, $\mathcal{F}_{\sigma'}$ is empty when the contacts in σ' are too far apart. Algebraic methods [21] and interval analysis [72] are general-purpose models for detecting infeasibility, but are too slow to be used effectively during planning. We use two simple geometric methods (similar to the work in [14]) that prove transition infeasibility. In our experiments, these prune about 70% of candidate transitions with low computational overhead.

Footfall reachability pruning Using a precomputation of the maximum span between two links, we prune footfalls that are too far apart to be reached. Define points p_k on link k for every link, and precompute $D_{i,j} \geq ||p_i - p_j||$ over all pairs of links i and j, and all configurations. Suppose the planner is at stance σ, and wishes to reach candidate footfall f constraining link j. For each link i fixed by σ, let q_i be

the fixed position of p_i. Let q_j be position of p_j fixed by f. Then, if $||q_j - q_i|| > D_{i,j}$ for any q_i, f is unreachable from σ. Furthermore, if the intersection of balls

$$B(q_j, D_{j,k}) \cap \left(\bigcap_i B(q_i, D_{i,k}) \right)$$

is empty for any k, then p_k cannot be reached simultaneously given the footfalls in σ and f. If either case, f can be pruned (Figure 3.7(c)). More accurate workspace bounds could improve this approach.

Center of mass reachability pruning Some footfalls cannot be reached while maintaining the center of mass inside the support polygon of σ. Perform a precomputation similar to before, but compute the upper bounds $\bar{D}_{i,j}$ on the distance between the center of mass CM_i of link i and p_j, for all i and j. Then, the robot's overall center of mass relative to p_j is contained within the ball $B(p_j, r)$, where

$$r = \frac{\sum_i m_i \bar{D}_{i,j}}{\sum_i m_i}$$

with m_i denoting the mass of link i. Then, if the footfall f fixes p_j at q_j, we check if the horizontal projection of $B(q_j, r)$ intersects the support polygon (Figure 3.7(d)). If they do not intersect, then f cannot be reached without losing equilibrium, and can be pruned.

3.3.6 Search Heuristics

In the transition-sampling search, the planner uses a priority queue to order the search by the most promising transitions. Consider a potential transition from an existing node σ to σ'. To improve the quality and speed of planning, we pick the transition that minimizes the cost function $g(\sigma) + h(\sigma, \sigma')$. We define g as the number of steps from the start stance to σ, and h as a weighted sum of several criteria:

- *Distance to goal.* Cost is increased proportionally to the distance between the centroid of the footfalls in σ' and those of the goal stance. This helps guide the

search toward the goal.

- *Footfall distribution.* Cost is increased proportional to the difference (in a least-squares sense) between the footfalls of σ' and those of a nominal stance on flat ground (with footfalls directly under each hip). This helps the planner select stances which are likely to be feasible.

- *Equilibrium criteria.* Cost is increased inversely proportionally to the area of the support polygon of σ'. This guides the search toward stances with a high margin of stability.

- *Sampling time.* Cost is increased as more time is spent sampling $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. This is particularly important, because it avoids spending too much time on potentially infeasible transitions. The planner has imperfect information about the feasibility of the transition. Failed samples suggest that $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ is empty or very small, and alternatives should be explored.

The sampling time heuristic is a "fuzzy" strategy that distributes computation among uncertain alternatives [101]. The merits of this approach will be discussed further in Section 5.6.

3.4 Experimental results

Experiments were performed in simulation on ATHLETE and HRP-2, and on the real Capuchin robot on a variety of real and simulated terrains. Most planning queries take in the range of minutes to an hour to solve on a 2GHz PC. This seems expensive, but the planner could be easily parallelized to achieve interactive rates [1].

3.4.1 Experiments on Athlete

Experiments show that the planner enables ATHLETE to walk across varied terrain, where fixed gaits would fail. A usability study shows that the planner is faster than a human in solving moderate to difficult problems.

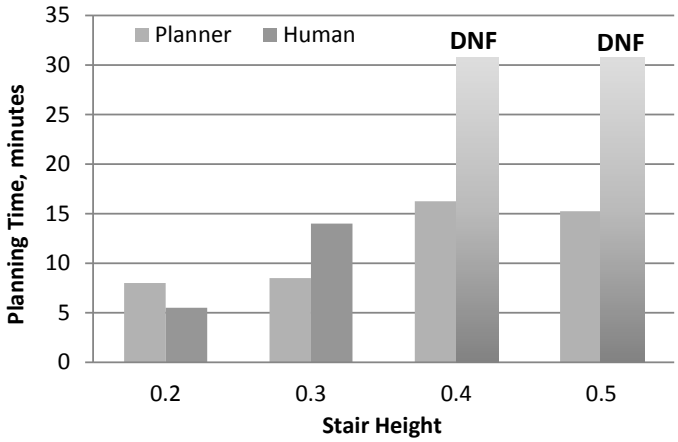


Figure 3.8: Comparing the planner to a human operator on stair steps of various heights. DNF indicates failure. Planner times are averaged over four runs.

Comparison with human operator I compared the planner against a human operator on a set of stair steps, with height ranging from 0.2 to 0.5 times the diameter of ATHLETE's chassis. On each problem, the planner used 200 randomly sampled candidate footfalls. The operator used a point-and-click interface to pick footfalls, with a PRM planner automatically planning single-step motions to reach those footfalls. Upon success, the operator immediately receives visual feedback, and is given the choice to accept the step, replan the motion, or backtrack to explore different footfall choices.

Figure 3.8 shows the timing results. The planner was able to plan consistently for each stair. Figure 3.9 shows an example of a motion climbing the 0.5-unit stair. Manual operation was straightforward for the 0.2-unit stair, but the 0.3-unit stair required a large amount of trial-and-error and backtracking. An attempt to plan the 0.4-unit stair was terminated in frustration after about 30 minutes.

The planning times correspond, in some sense, to an intuition of how "difficult" a query seems. Consider a plan for the 0.3- and 0.5-unit stairs in more detail. For the 0.3-unit stair, an average of about 5 minutes were spent in transition-sampling search, and the remaining 3 were spent in single-mode PRM planning. Of the billions of possible stances, the search sampled in about 2,500 transition regions, reached 200 feasible stances, and attempted single-step planning in about 60. For the 0.5-unit

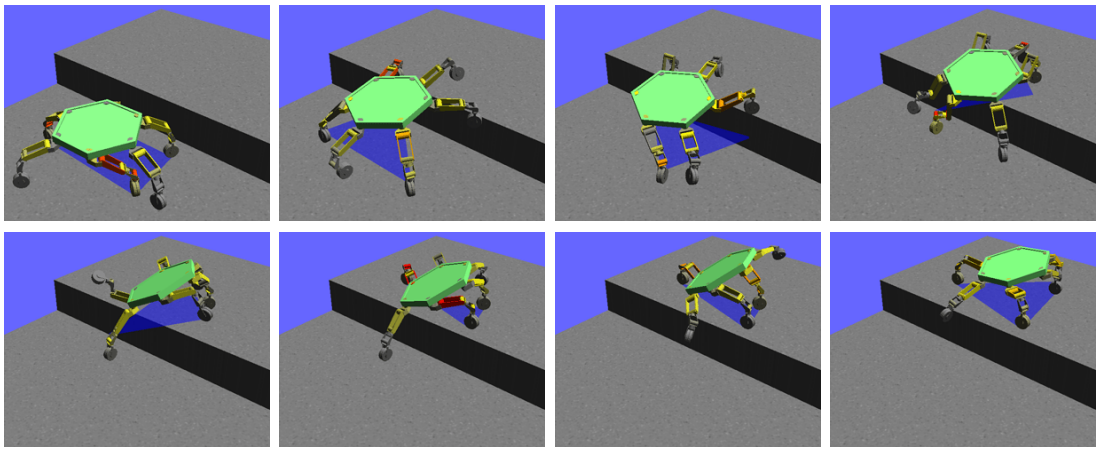


Figure 3.9: Climbing a 0.5-unit stair step.

stair, a similar amount of time was spent in single-mode planning, but time spent in transition-sampling search was almost tripled. This suggests that footfalls must be chosen more carefully when climbing the higher stair than the lower one.

Lunar-like terrain Experiments were performed on a set of surfaces generated to resemble a variety of lunar terrain. Each terrain model is a height-map of the form $z = f(x, y)$, created using a fractal generation method and represented by a triangular mesh consisting of 32768 triangles, each about the size of one of ATHLETE's wheels.

Figure 3.10 shows motion on smooth, undulating ground (where all contacts are modeled with the same coefficient of friction). The initial and final stances are at a distance of about twice the radius of ATHLETE's chassis. The resulting motion consisted of 66 steps. Total computation time was 14 minutes. Figure 3.11 shows motion on irregular and steep ground. The resulting motion consisted of 84 steps. Total computation time was 26 minutes. For comparison, Figure 3.12 shows the result of applying a common fixed gait (an alternating-tripod) to both of these terrains. On smooth ground, the gait works well – it requires little computation, and results in more efficient motion (Figure 3.12a). On irregular and steep ground, however, the gait does not work at all – it causes ATHLETE to lose balance or exceed torque limits at several locations (the configurations shown in red in Figure 3.12b).

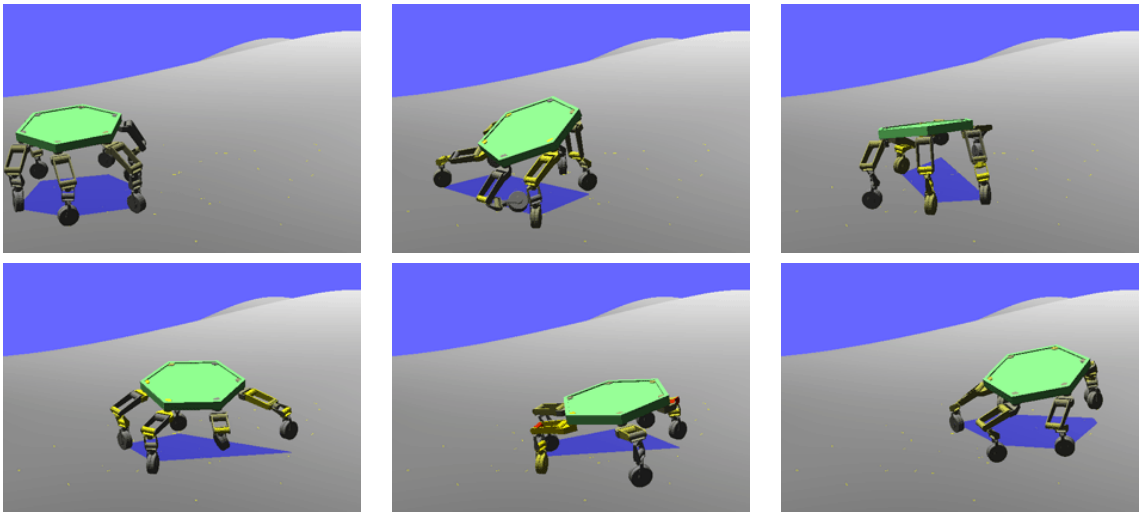


Figure 3.10: Walking on smooth, undulating terrain with no fixed gait.

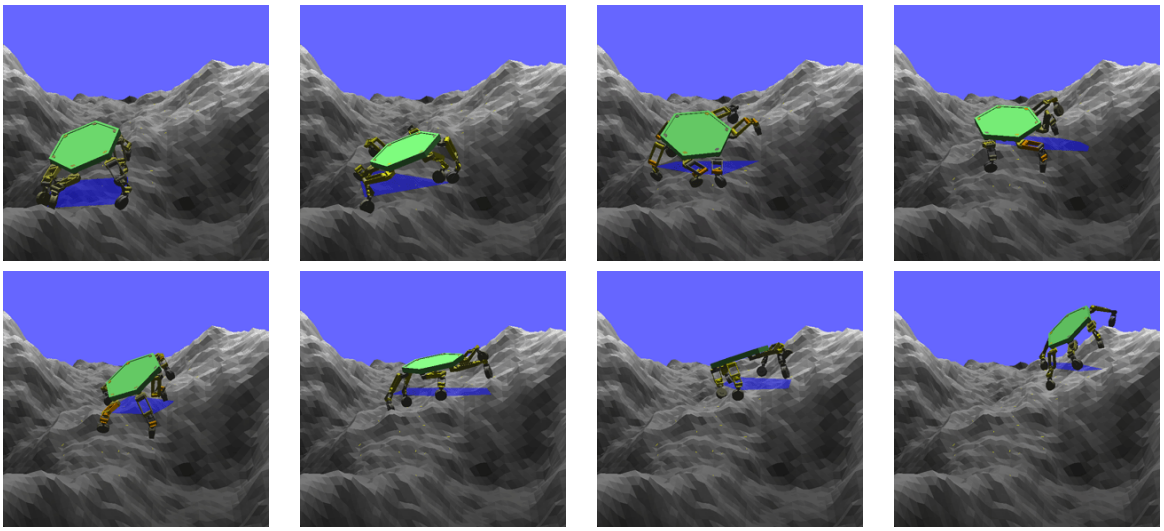
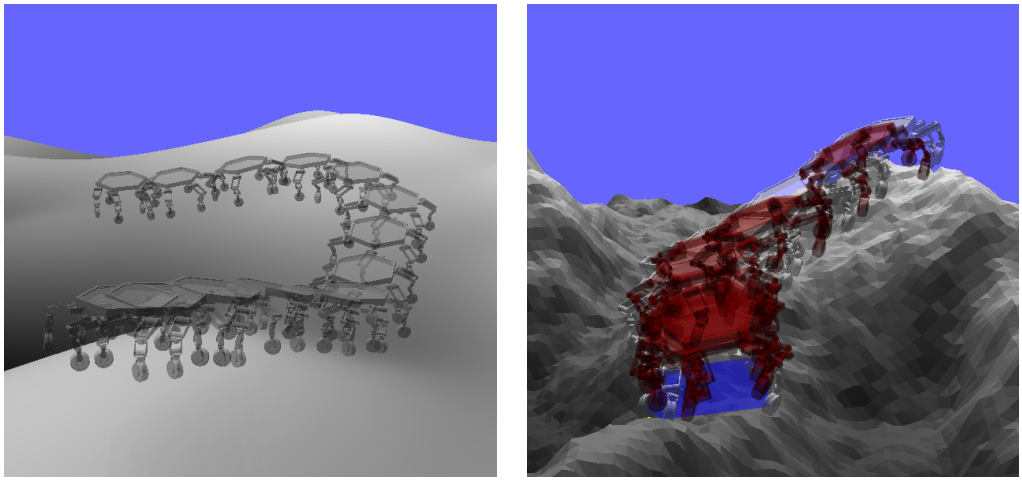


Figure 3.11: Walking on steep, uneven terrain with no fixed gait.



(a) (b)

Figure 3.12: Walking with an alternating tripod gait is (a) feasible on smooth terrain but (b) infeasible on uneven terrain. Infeasible configurations are highlighted red.

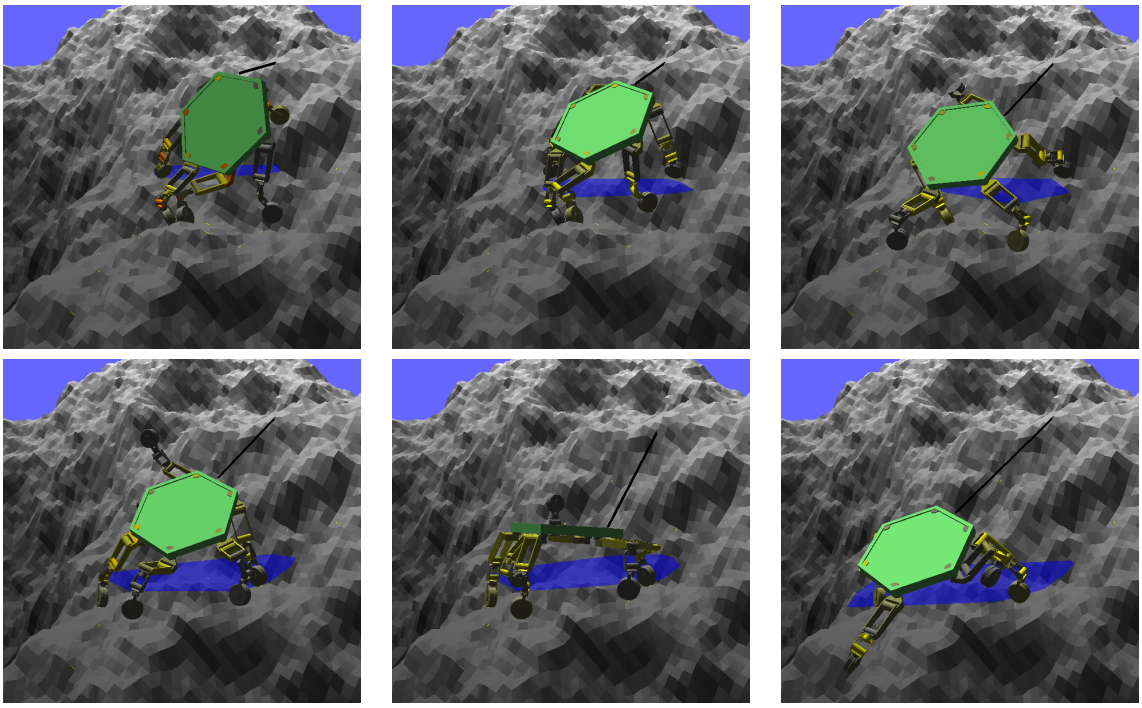


Figure 3.13: Rappelling down an irregular 60° slope with no fixed gait.

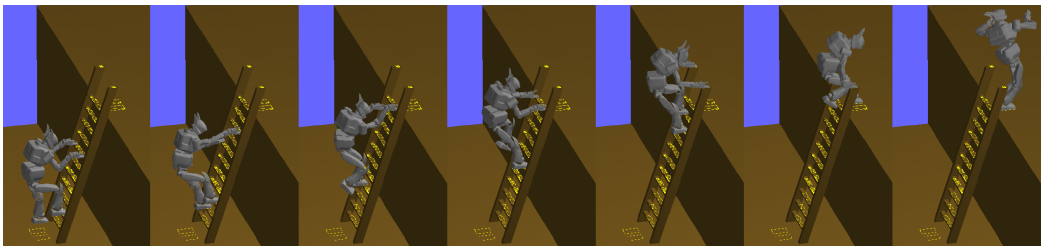


Figure 3.14: Climbing a ladder.

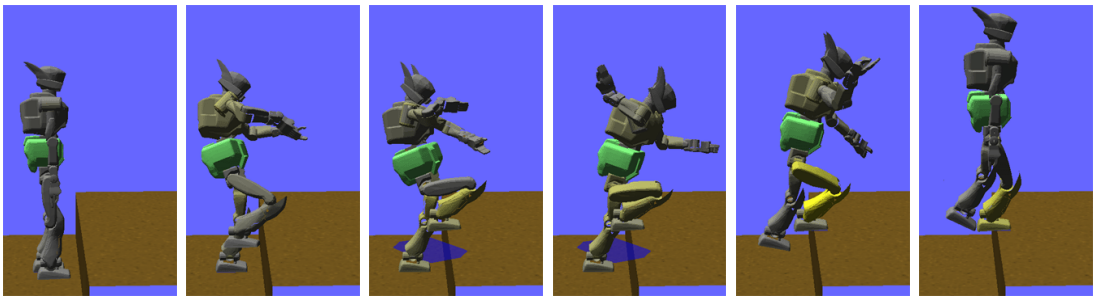


Figure 3.15: A 0.4m stair step that can be climbed only using feet.

Figure 3.13 shows motion to descend irregular and steep terrain at an average angle of about 60°. In this example, ATHLETE is rappelling, using a tether (anchored at the top of the cliff) to help maintain equilibrium. The tether is included with no modification to the planner, treating it as an additional leg with a prismatic telescopic joint[1]. The resulting motion consisted of 32 steps. Total computation time was 16 minutes.

3.4.2 Experiments on HRP-2

The planner was tested on the HRP-2 humanoid robot in simulation. Figure 3.14 shows a ladder climbed with hands and feet. Here the planner used 45 candidate hand and footfalls placed manually. Total planning time was approximately 3 hours, most of which was spent in single-mode planning.

The planner was able to discover several HRP-2 inherent capabilities that were

[1]Here, the tether is modeled by a long rod of variable length and unidirectional force. Modeling it as a deformable object with interactions with the terrain would be much more complicated.

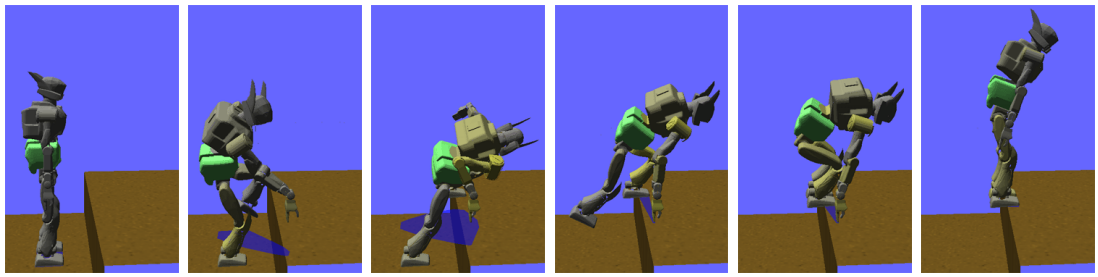


Figure 3.16: A 0.5m stair step that requires a hand to aid stability.

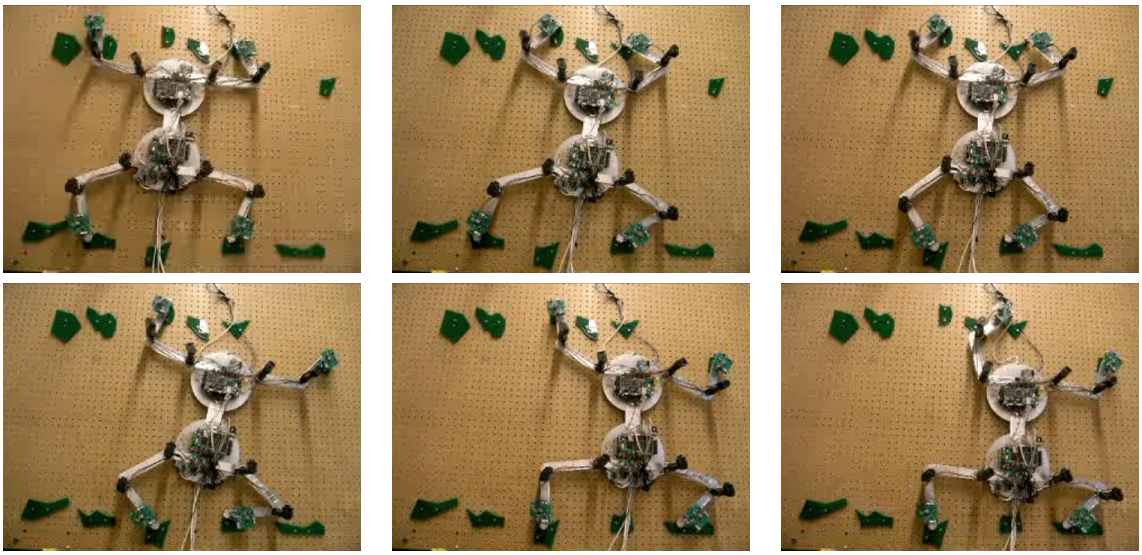


Figure 3.17: A five step motion performed by the Capuchin climbing robot.

previously unknown. For example, a 0.4m stair step could be traversed just using its feet (Figure 3.15), but a 0.5m stair step requires the use of hands to remain stable (Figure 3.16).

3.4.3 Experiments on the Capuchin Climbing Robot

Experiments were performed on Capuchin moving laterally on a near vertical wall using a sparse set of protruding, irregularly shaped "holds". Here, candidate contacts are easily identified, so planning is very fast (less than a minute). Figure 3.17 displays frames from an execution of the planned path, using a controller that tracks the body

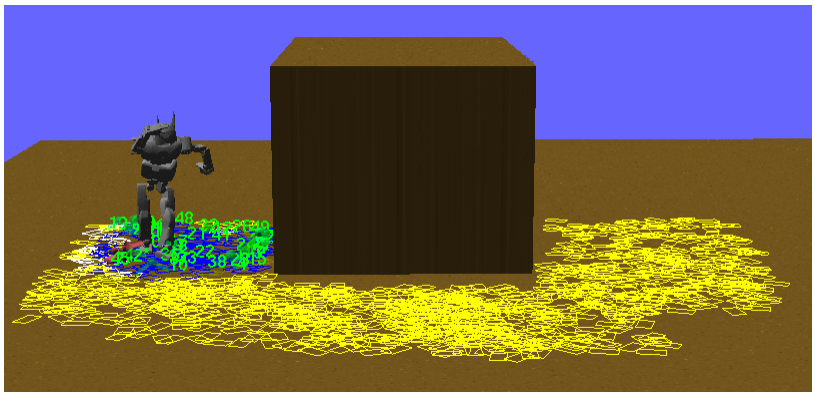


Figure 3.18: After 1000 transitions are sampled, only the footsteps colored in blue are reached.

trajectory while balancing the forces applied at the contact points [97]. The entire motion was executed in 30 seconds.

3.5 Discussion

The mode-before-motion planner presented in this chapter has been shown to be capable of producing complex motions for a wide variety of robots on highly irregular terrain. It can be easily adapted to different legged robot morphologies, making it particularly useful for exploring the capabilities of robot designs in simulation. However, the planner has several weaknesses, some of which will be addressed in later chapters.

First, the motions are often unnatural, which is most troublesome in humanoid robots. For example, the arms flail while walking bipedally. This is caused by both randomized sampling in single-mode PRM planning and randomized sampling of transition configurations. Furthermore, the planner may choose to use hand or knee contacts to traverse flat ground, because contacts are sampled randomly as well. The next chapter will address how to improve the quality of motions while still remaining general.

Second, the planner sometimes works slowly even when solutions are obvious. Transition-sampling search relies solely on a heuristic to provide global guidance. If

the heuristic contains local minima, the search will spend a lot of time exhaustively exploring around minima. For example, consider a problem where HRP-2 is asked to reach a goal on the other side of an obstacle (Figure 3.18). To a human, the solution of walking around the obstacle is plainly obvious. After 1000 transitions are explored, and about 10 minutes of computation, the planner has not yet even begun to explore around the obstacle. This problem can be avoided by guiding the planner with coarse planning [36], better heuristics, or human-specified waypoints.

Third, parameter tuning is expensive. For example, consider selecting the weight for the equilibrium region search heuristic (Section 3.3.6). The size of support polygons can be smaller for bipedal robots (which routinely balance on one foot) than for a six-legged robot like ATHLETE, and this condition should be weighted accordingly. But when gathering data to assess the performance of one set of weights, many queries are needed to 1) reduce the variance caused by randomness and to 2) avoid tuning to a small number of examples. Each multi-step planning query takes several minutes, making extensive testing prohibitively expensive. The decision theoretic approach of Chapter 5 provides an alternative approach. This approach gathers data about the performance of subproblems (which is much faster than gather data about the whole problem) to help compose better strategies. It also uses deliberate reasoning, which reduces the reliance on art when selecting parameters.

Finally, the planner works best when many sequences of steps lead to the goal (in which case the mode graph search will find a sequence quickly), or useful contacts can be inferred from terrain properties (like slope, roughness, or friction which can aid the selection of candidate contacts). But it has difficulty when only carefully chosen steps can reach the goal, and these steps cannot be inferred from terrain properties. For example, consider the problem of Figure 3.16, where HRP-2 climbs a 0.5m stair step. The hand must be placed in a precise location — a few centimeters to either side, or a twist of a few degrees would not allow the robot to climb the step. If the set of candidate contacts does not contain such a placement, the planner will fail. In some sense, they can be considered "bottlenecks" in space. These planning queries seem hard in a general sense, as they cannot be easily distinguished from an infeasible query, and would likely prove difficult for a human operator. If such queries

need to be solved often, perhaps the planner could use past experience to learn better exploration strategies. This would be a promising area of future work (see Chapter 6).

Chapter 4

Using Domain Knowledge in Planning

The planner in the previous chapter is largely restricted to using information gathered by testing constraints. This is advantageous when little else is known about robot's capabilities. But prior information about the system, *domain knowledge*, can be used to the planner's advantage. Predefined short-term behaviors, such as walking gaits or repeatable motion clips, are commonly used. Restricting the planner to sequence and compose a small set of high-quality behaviors reduces the search space. This improves motion quality and makes planning fast. But these benefits come at the expense of reliability. Behavior sequencing may work for some problems, but it is impractical to engineer enough behaviors to navigate irregular terrain or manipulate irregular objects. Even if this were practical, it would nullify the potential speed benefits because the planner would be overwhelmed by choice. This chapter presents two types of domain knowledge that help improve planning without sacrificing reliability.

The first technique uses *motion primitives*, high quality motion clips, as "example motions" to improve locomotion planning (particularly for humanoid robots). A primitive is used as domain knowledge that suggests that similar motions will also be high quality. For example, consider a natural motion taking a step on flat ground. A similar motion that takes a step on stairs will not be exactly the same, but it will still be natural. Given a pre-defined primitive, the planner derives a *sampling strategy*

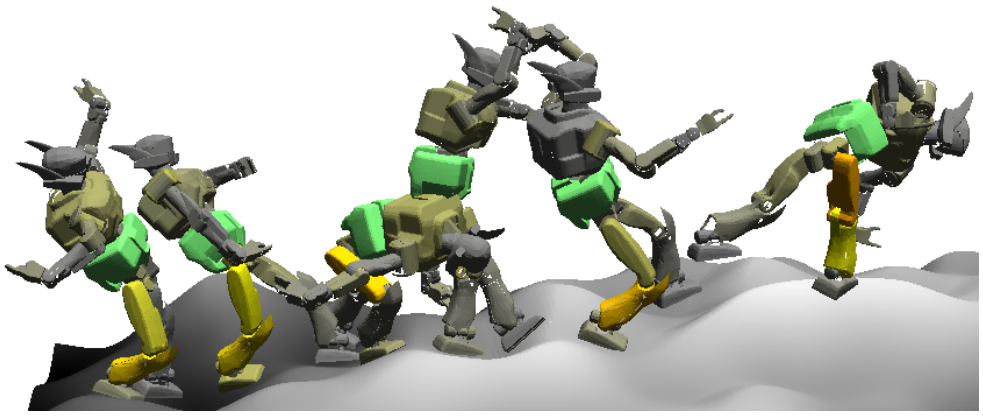


Figure 4.1: A jerky, unnatural motion produced by random sampling

that biases its selection of contacts and single-step motions. This method generates high-quality motions even on irregular terrain, and provides a modest computational speedup.

The second technique improves the speed and quality of a planner for nonprehensile (pushing) manipulation on the ASIMO humanoid robot. ASIMO must stand while pushing, because its hand cannot be positioned accurately while it walks. Its reach is limited, so it must repeatedly walk and push to move object any reasonable distance (say, across a table). The planner faces a challenge in choosing where to stop walking and start pushing. To execute a high-utility push (one that moves the object a long distance in a given direction), ASIMO must precisely choose where it stands. Otherwise, it has to use several shorter pushes, repositioning itself each time, to move the object the same distance. Here, we precompute a utility estimate that measures how far an object can be expected to be pushed. In planning, this estimate helps sample mode switches to yield higher utility pushes. This *utility-centered expansion* approach greatly improves planning speed and quality, to the point where motions can be planned and executed on ASIMO at interactive rates.

4.1 Motion Primitives

Multi-modal planners that use random sampling often produce motions of low quality (see Figure 4.1). They may contain unnatural jerks, which may waste energy, waste time, and distract human operators. This latter issue is especially troublesome for humanoid robots like HRP-2, which are designed to operate like humans. Variational [12] and "shortcutting" techniques [48] can smooth motions in postprocessing. But it is difficult to improve a motion in postprocessing when its poor quality is caused by an unnatural choice of footfalls.

This section presents a technique, originally presented in [55], that combines the motion library approach with a randomized approach. The planner uses a predefined library of *motion primitives*, which are high-quality motion clips. The planner selects a motion primitive to use as an example, and *biases* its selection of footfalls and single-mode paths to be near the example path.

First, we describe how we define and generate a library of primitives. Then, we will describe how they guide our selection of paths, transitions, and footfalls. Experiments show that a wide variety of terrain can be traversed, even with just a few primitives. This improves the speed and quality of the sample-based planner while retaining much of its generality.

4.1.1 Related Work

Motion primitives, maneuvers, and other types of macro-actions have been applied widely to robotics and digital animation. Four general strategies have been used:

Record and playback. This strategy restricts motion to a library of maneuvers. Natural-looking humanoid locomotion on mostly flat ground can be planned as a sequence of precomputed feasible steps [83]. Robust helicopter flight can be planned as a sequence of feedfoward control strategies (learned by observing skilled human operators) to move between trim states [44, 46]. Robotic juggling can be planned as a sequence of feedback control strategies [31]. The motion of peg-climbing robots can be planned as a sequence of actions like "grab the nearest peg" [10]. In these

applications, a reasonably small library of maneuvers is sufficient to achieve most desired motions. For humanoid robots on varied terrain, such a library may grow to impractical size.

Warp, blend, or transform. Widely used for digital animation, this strategy also restricts motion to a library of maneuvers, but allows these maneuvers to be superimposed or transformed to better fit the task at hand. For example, captured motions of human actors can be "warped" to allow characters to reach different footfalls [124] or "retargetted" to control characters of different morphologies [50]. Of course, for a digital character it is most important to look good while for a humanoid robot it is most important to satisfy hard motion constraints. So although some techniques have been proposed to transform maneuvers while maintaining physical constraints [109, 116], this strategy seems better suited for animation than robotics.

Model reduction. This strategy plans overall motion first, following this motion with a concatenation of primitives. For example, another way to generate natural-looking humanoid locomotion on flat ground is to approximate the robot as a cylinder, plan a 2-D collision-free path of this cylinder, and follow this path with a fixed gait [77, 78, 80, 107]. A similar method is used to plan the motion of nonholonomic wheeled vehicles [84, 85]. A related strategy plans the motion of key points on a robot or digital actor (such as the center of mass or related ground reference points [108]), tracking these points with an operational space controller [115]. These approaches work well when it does not matter much where a robot or digital actor contacts its environment. When the choice of contact location is critical, as is often the case for humanoids on varied terrain, it makes more sense to compute a sequence of footfalls first.

Bias inverse kinematic solutions. Like model reduction, this strategy first plans the motion of key points on a robot or digital actor, such as the location of hands or feet. But instead of a fixed controller, a search algorithm is used to compute a pose of the robot or actor at each instant that tracks these points (an inverse kinematic

solution). One approach is to choose an inverse kinematic solution according to a probability density function learned from high-quality example motions [54, 92, 96, 127]. The set of examples give the resulting pose a particular "style." The motion primitive approach is similar, with a sampling strategy that picks contacts and configurations in a neighborhood of the primitive.

4.1.2 Definition

We record each primitive as a nominal path

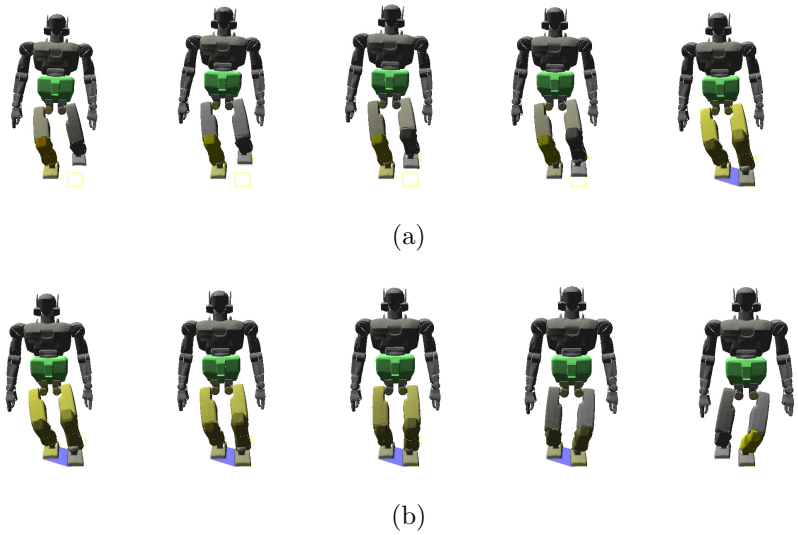
$$u\colon t \in [0,1] \to u(t) \in \mathcal{Q}$$

in configuration space that moves from stance σ to σ'. In particular, it does one of two things:

- *Adds a contact.* For some σ and σ' such that $\sigma \subset \sigma'$, u is a feasible path in $\mathcal{F}_\sigma$ from $u(0) \in \mathcal{F}_\sigma$ to $u(1) \in \mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$.
- *Removes a contact.* For some σ and σ' such that $\sigma \supset \sigma'$, u is a feasible path in $\mathcal{F}_\sigma$ from $u(0) \in \mathcal{F}_\sigma$ to $u(1) \in \mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$.

We will denote the start and goal stances for each primitive u by σ_u and σ'_u, respectively.

The motion of u is only feasible when moving from σ_u to σ'_u, and not between other stances. But feasibility is (mostly) invariant under certain transformations of u, σ_u, and σ'_u. In particular, any arbitrary translation and rotation about the gravity vector does not violate constraints (collision with the environment excepted). Invariance is a key reason why a small number of steps can traverse large areas of flat terrain. It is also part of the intuitive notion of what it means for two steps to be "similar" – not that they occur in similar locations in space, but rather, they make similar relative changes in stance.



(a)

(b)

Figure 4.2: Two primitives on flat ground, to (a) place a foot and (b) remove a foot. The support polygon – here, just the convex hull of supporting feet – is shaded blue.

4.1.3 Generating a Library of Primitives

Currently, it is the responsibility of the user to generate primitives to include in the library. First, the user should identify a small but representative set of steps to include. These steps should be both important (often repeated) and broadly applicable (similar to a wide variety of other steps). For example, we might choose to include several consecutive steps on flat ground, each placing or removing a foot (Figure 4.2).

Next, the user should generate high-quality motions to achieve each step. The notion of quality is deliberately vague, and could be any characteristics desired of the robot's motion. If these characteristics are quantified (e.g, minimizing path length, torque, energy, or the amount of deviation from an upright posture), motion generation can be automated as follows. First, we generate an initial trajectory between the given start and goal stances by randomly sampling a feasible transition and creating a path to reach it using PRM, as in [57, 19]. Then, we optimize this trajectory with respect to the given objective function using a standard nonlinear optimization package [89]. This entire process is an off-line precomputation; several hours were required to generate the two example primitives in Figure 4.2.

The generation of motion primitives has not been the main focus of this work, so

many improvements may be possible. For example, we expect the optimization step could be accelerated using the method of [12]. Optimization may not work if quality is hard to quantify, such as the aesthetic notion of looking "natural". Human motion capture datasets can be segmented to identify representative steps and natural motions [77]. Likewise, we might use a learned classifier to decide (without supervision) whether candidate primitives look natural, as in [111]. Finally, we might automate the selection of primitives to include in our library by learning a statistical model of importance (similar to location-based activity recognition [91]) or applicability after perturbation (similar to PRM planning with model uncertainty [98]).

4.1.4 Finding Paths

Motion primitives help the planner generate steps. As described in the following sections, we do this at three levels: finding a path (given a transition and a final stance), finding a transition (given only the final stance), and finding a contact (in order to define the final stance). In each case, first we transform the primitive to better match the step we are trying to plan, then we apply the transformed primitive to bias the planner's sampling strategy.

Consider the robot at an initial configuration $q_{\text{initial}} \in \mathcal{F}_\sigma$ at an initial stance σ. Assume that we are given a final stance σ' and a transition $q_{\text{final}} \in \mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$ (recall q_{final} is a configuration feasible at both σ and σ'). Also assume that we are given an appropriate primitive $u \subset \mathcal{Q}$. We want to use u to guide our search for a path from q_{initial} to q_{final} in $\mathcal{F}_\sigma$. As in Chapter 3, we use SBL to grow trees from root configurations (see Section 3.3.4). But rather than root these trees only at q_{initial} and q_{final}, we root them at additional configurations (similar to [1]) sampled according to the primitive u.

Transforming the primitive to match q_{initial} and q_{final}. Although we assume u is similar to the step we are trying to plan, it will not be identical. So first, we transform u so that it starts at q_{initial} and ends at q_{final}. We have chosen to use an

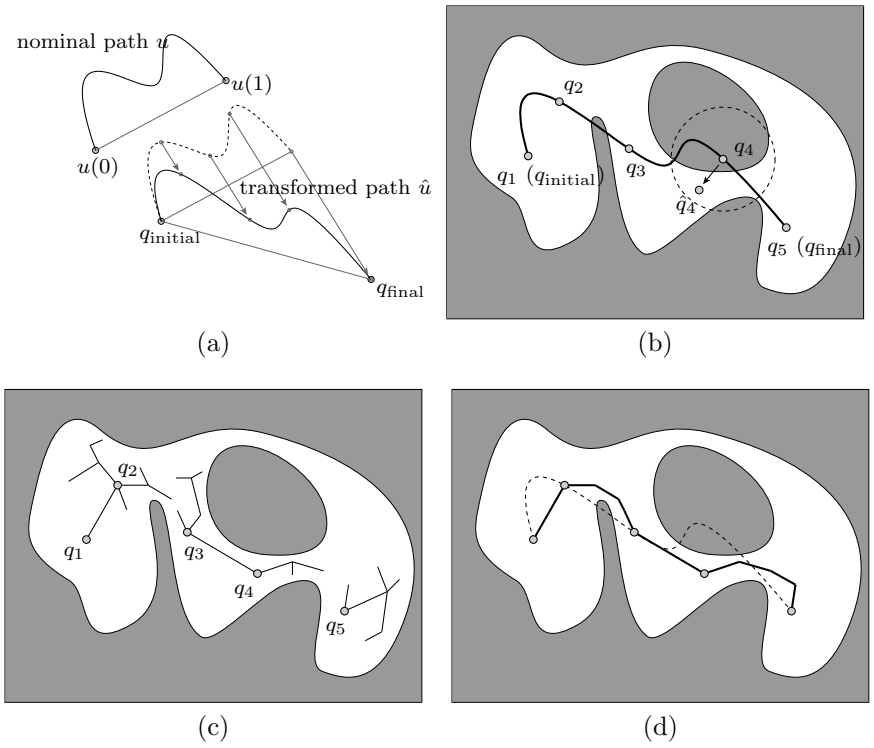


Figure 4.3: Using a primitive to guide path planning. (a) Transforming a motion primitive to start at q_{initial} and end at q_{final}. (b) Sampling root milestones in $\mathcal{F}_\sigma$ near equally spaced waypoints along $\hat{u}$. (c) Growing trees to connect neighboring roots. (d) The resulting path, which if possible is close to $\hat{u}$ (dotted).

affine transformation of the form

$$\hat{u}(t) = A\,(u(t) - u(0)) + q_{\text{initial}} \tag{4.1}$$

that maps the straight-line segment between $u(0)$ and $u(1)$ to the segment between q_{initial} and q_{final}. In other words,

$$\begin{aligned} \hat{u}(0) &= A\,(u(0) - u(0)) + q_{\text{initial}} \\ &= 0 + q_{\text{initial}} \\ &= q_{\text{initial}} \end{aligned} \qquad \begin{aligned} \hat{u}(1) &= A\,(u(1) - u(0)) + q_{\text{initial}} \\ &= (q_{\text{final}} - q_{\text{initial}}) + q_{\text{initial}} \\ &= q_{\text{final}} \end{aligned}$$

In particular, we select A closest to the identity matrix, minimizing

$$\min_A \sum_{i,j} (A_{ij} - \delta_{i,j})^2 \text{ such that } A\left(u(1) - u(0)\right) = q_{\text{final}} - q_{\text{initial}}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. A can be computed in closed form as

$$A = I + \frac{\left((q_{\text{final}} - q_{\text{initial}}) - (u(1) - u(0))\right)\left(u(1) - u(0)\right)^T}{\|u(1) - u(0)\|_2^2}.$$

This transformation is visualized in Figure 4.3a. First, u is translated to start at q_{initial}. Then, the farther we move along u (the more we increase t), the closer $\hat{u}$ is pushed toward the segment from q_{initial} to q_{final}.

Sampling root milestones. Let $q_1, \ldots, q_n$ be configurations evenly distributed along $\hat{u}$ from q_{initial} to q_{final} (Figure 4.3b). For each $i = 1, \ldots, n$, we test if $q_i \in \mathcal{F}_\sigma$. If so, we add q_i as a root milestone in our roadmap. If not, we repeatedly sample other configurations in a growing neighborhood of q_i until we find some feasible $q_i' \in \mathcal{F}_\sigma$, which we add as a root instead of q_i.

Connecting neighboring roots with sampled trees. For $i = 1, \ldots, n-1$, we check if the root milestone q_i can be connected to its neighbor q_{i+1} with a feasible local path (as in Section 3.3.4). If not, we add the pair of roots (q_i, q_{i+1}) to a list $\mathcal{R}$. Then, we apply SBL to grow trees between every pair in $\mathcal{R}$. For example, in Figure 4.3c we add (q_2, q_3) and (q_4, q_5) to $\mathcal{R}$ and grow trees to connect both q_2 with q_3 and q_4 with q_5. We process all trees in parallel. So at every iteration, for each pair $(q_i, q_{i+1}) \in \mathcal{R}$, we first add m milestones to the trees at both q_i and q_{i+1} (in our experiments, we set $m = 5$). Then, we find the configurations q connected to q_i and q' connected to q_{i+1} that are closest. If q and q' can be connected by a local path, we remove (q_i, q_{i+1}) from $\mathcal{R}$. When we connect all neighboring roots, we return the resulting path; if this does not happen after a fixed number of iterations, we return failure. Just like our original implementation, this approach will find a path between q_{initial} and q_{final} whenever one exists (given enough time). However, since we

seed our roadmap with milestones that are close to u, we expect the resulting motion to be similar (and of similar quality) to this primitive whenever possible (Figure 4.3d), deviating significantly from it only when necessary.

4.1.5 Finding Transitions

Again consider the robot at a configuration $q_{\text{initial}} \in \mathcal{F}_\sigma$ at a stance σ. But now, assume that we are only given a final stance σ', so we use a primitive u to guide our search for a transition before we plan a path to reach it.

Transforming the primitive to match σ and σ'. Since we do not know q_{final}, we can not use the same transformation (4.1) that we used for planning paths. Instead, we choose a rigid-body transformation of the form

$$\hat{u}(t) = Au(t) + b \tag{4.2}$$

that maps the nominal stances σ_u and σ'_u (associated with the primitive u) as closely as possible to the stances σ and σ'.

Recall that a stance consists of several contacts, each placing a link of the robot on the terrain. We can define the location of each placement by a finite number of points $r_i \in \mathbb{R}^3$. For example, the face-face contact between a foot and the ground might be defined by the vertices r_1, r_2, and r_3 of a triangle. We consider these points to be attached to the robot, so if the foot is placed against a different face in the terrain, the points r_1, r_2, and r_3 move in $\mathbb{R}^3$ but remain in the same location relative to the foot. We will use these points to define our mapping between stances.

In particular, let $r_i \in \mathbb{R}^3$ for $i = 1, \ldots, m$ be the set of all points defining the contacts in both σ_u and σ'_u, and let $s_i \in \mathbb{R}^3$ for $i = 1, \ldots, m$ be the set of all points defining the contacts in both σ and σ'. (We assume u has been chosen so that both sets have the same number of points.) Then we choose the rotation matrix A and translation b in (4.2) that minimize

$$\min_{A,b} \sum_i \|Ar_i + b - s_i\|_2^2.$$

We can compute A and b in closed form [5]. But, we only consider rotations A about the gravity vector to avoid tilting the robot into an unstable orientation.

Sampling a transition. As before, we sample configurations $q \in \mathcal{Q}_\sigma$, keeping them if $q \in \mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'}$. But rather than sample configurations completely at random, we sample them in a growing neighborhood of $\hat{u}(1)$. We expect a well-chosen transition to further improve the quality of the path to reach it.

4.1.6 Finding Footfalls

Once more, consider the robot at a configuration $q_{\text{initial}} \in \mathcal{F}_\sigma$ and a stance σ. But now, assume we are given neither a final stance nor a transition, but only a primitive u. If u removes a robot link from the terrain, we immediately generate a final stance σ' by removing the corresponding footfall from σ. But if u places a link in the terrain, we use it to guide our search for a new footfall.

Transforming the primitive to match σ. We use the same transformation (4.2) to construct $\hat{u}$ as for finding transitions. But here, we compute A and b to map only σ_u to σ, since we do not know σ'. We use this transformation to adjust the placement of the new footfall given by u. Let $r_i \in \mathbb{R}^3$ for $i = 1, \ldots, m$ be the set of points defining this contact. Then the transformed contact is given by $\hat{r}_i = Ar_i + b$ for $i = 1, \ldots, m$.

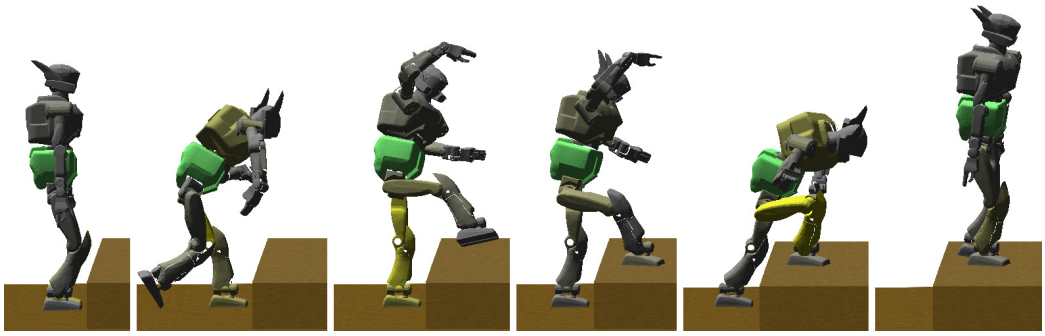
Sampling a footfall. This transformed contact is adjusted further to match the terrain surface. We define a sphere of radius δ, centered at $(1/m)\sum_i \hat{r}_i$. We increase δ until the intersection of this sphere with the terrain is non-empty (initially, we set δ approximately the size of HRP-2's foot). We randomly sample a placement of the points $\hat{r}_i$ on the surface of the terrain inside the sphere, by first sampling a position of their centroid $s \in \mathbb{R}^3$ on the terrain surface, then sampling a rotation of $\hat{r}_i$ about the surface normal at s. We check that the footfall defined by this placement has similar properties (normal vector, friction coefficient) to the footfall defined by u.

If so, we add it to σ to form σ'. If not, we reject it, grow the neighborhood δ, and sample another placement.

4.1.7 Examples

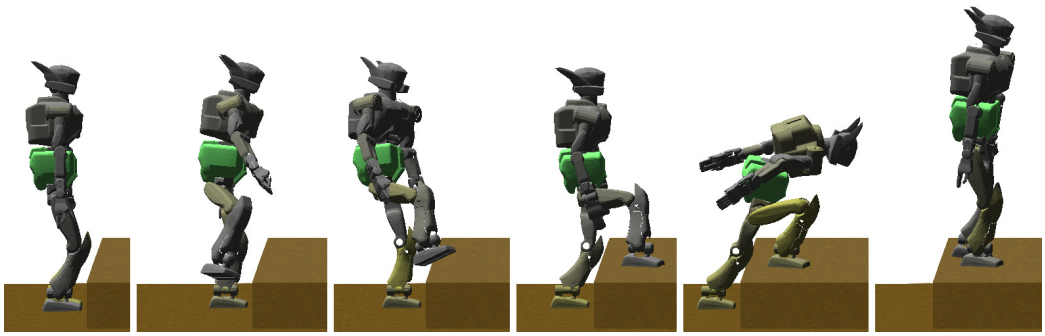
An example of climbing a single stair. With each additional part of a step that we compute using a primitive, we add to the quality of the result. For example, consider the motion of HRP-2 in Figure 4.4a to climb a single stair of height 0.3m (just below the knee). This motion was planned from scratch, by randomly sampling contacts and transitions and by using PRM to generate paths. The robot does not look natural – its arm and leg motions are erratic, and its step over the stair is needlessly long. To improve this motion, we applied the two primitives shown in Figure 4.2 (steps on flat ground). Figure 4.4b shows the result of using these primitives to plan each path. Some erratic leg motions are eliminated, such as the backward movement of the leg in the second frame. The erratic arm motions remain, however, because the transition in the fourth frame is the same (still randomly sampled). Figure 4.4c shows the result of using primitives to adjust this transition as well as to plan paths, eliminating most of the erratic arm motions. Finally, Figure 4.4d shows the result of using primitives to select footfalls well as plan transitions and paths. The chosen footfall resulted in a much easier step, eliminating the extreme lean in the fifth frame.

Planning time and motion quality for stairs of different heights. In our experiments, we have observed that planning time remains low and motion quality remains high even when we use a primitive to plan a step that is quite different. For example, we adapted the same two primitives in Figure 4.2 to stairs of height 0.2m, 0.3m, and 0.4m. Figure 4.5 shows the results, averaged over five runs. Quality is measured by an objective function that penalizes both path length and deviations from an upright posture (lower values indicate higher quality). For comparison, we report the minimum objective value achieved after a lengthy off-line optimization. These results demonstrate that our use of primitives provides a modest reduction in planning time but significantly improves motion quality. Note also that both time and quality degrade gracefully as the step we are planning deviates further from the

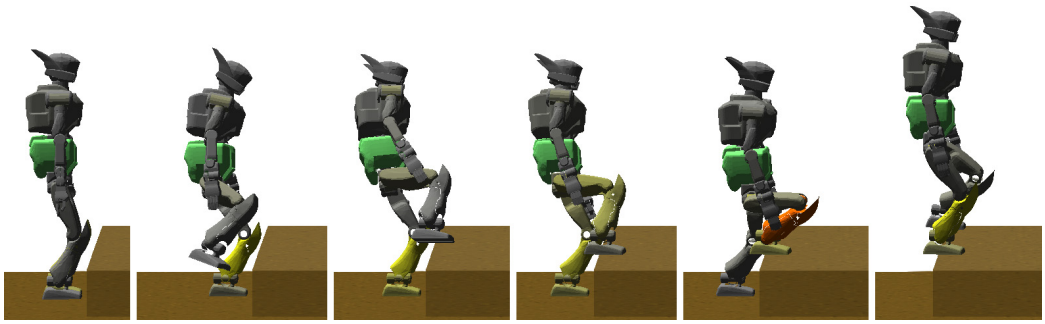


(a) Stair step planned entirely from scratch.

(b) Primitives guide path planning, reducing unnecessary leg motions.

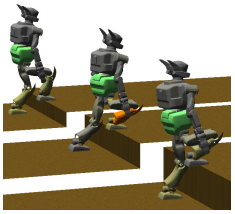


(c) Primitives guide transition sampling, reducing unnecessary arm motions.



(d) Primitives guide the choice of contact, resulting in an easier step.

Figure 4.4:



Stair height	From scratch		Adapt primitive		Optimal objective
	Time	Objective	Time	Objective	
0.2m	8.61	5.03	5.42	3.04	2.19
0.3m	10.3	4.67	4.08	2.31	2.17
0.4m	12.2	5.15	10.8	3.27	2.55

Figure 4.5: Planning time and objective function values for stair steps, averaged over 5 runs.

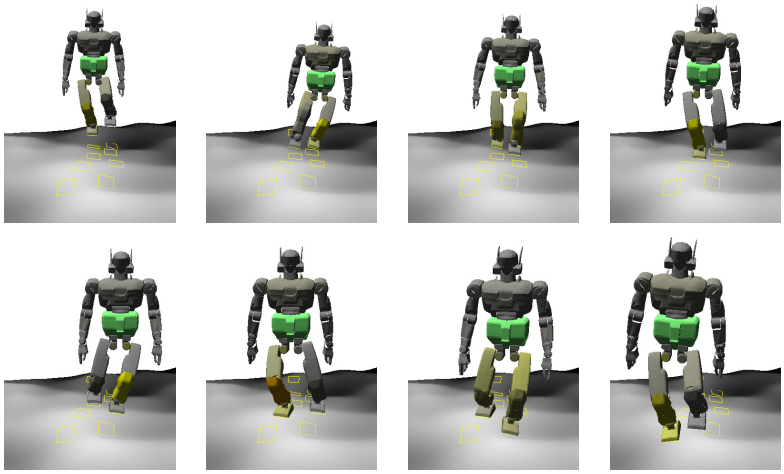


Figure 4.6: A planar walking primitive adapted to slightly uneven terrain.

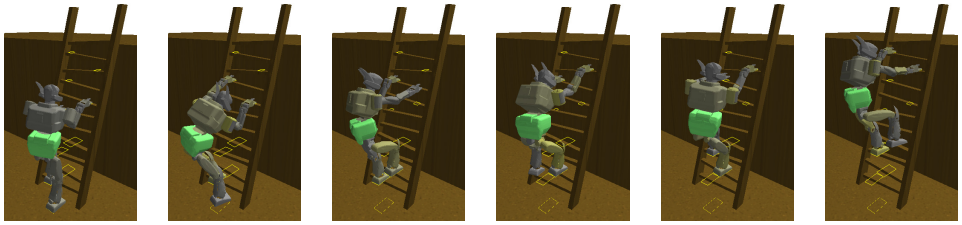


Figure 4.7: A ladder climbing primitive adapted to a new ladder with uneven rungs.

primitive.

Motions Generated by Multiple Primitives. Finally, I show several examples where primitives are applied repeatedly to generate a longer motion. Figure 4.6 shows HRP-2 on uneven terrain (using the primitives in Figure 4.2), in which the

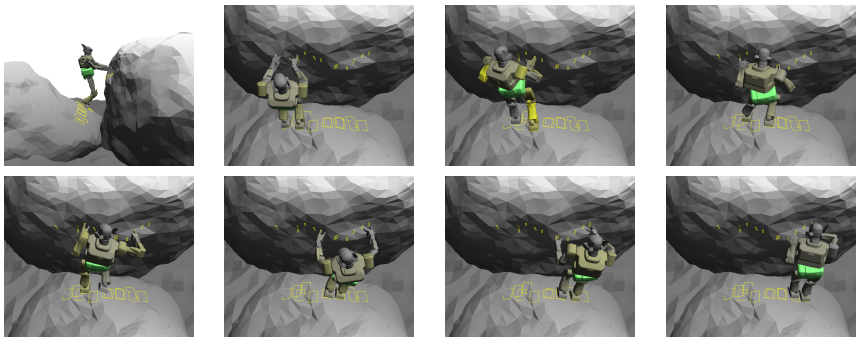


Figure 4.8: A side-step primitive using the hands for support, adapted to a terrain with large boulders. Hand support is necessary because the robot must walk on a highly sloped boulder.

Figure 4.9: A motion on steep and uneven terrain generated from a set of several primitives. A hand is being used for support in the third configuration.

highest and lowest point differ by 0.5m. Figure 4.7 shows HRP-2 climbing a ladder with rungs that have non-uniform spacing and that deviate from horizontal by up to 15°. The primitives for this example were generated on a ladder with horizontal, uniformly spaced rungs. Figure 4.8 shows HRP-2 making several sideways steps among boulders, using the hands for support. Here, the primitives were generated by stepping sideways on flat ground while pushing against a vertical wall. Figure 4.9 shows HRP-2 traversing very rough terrain with slopes up to 40°. This motion was generated with a larger set of primitives (including steps of several heights, a pivot step, and a high step using the hand for support). In all of these examples, contacts

were sampled on-the-fly (using motion primitives), not placed by hand. Planning for the first three examples took about one minute on a 1.8 GHz PC. The fourth took example about eight minutes.

4.1.8 Selecting Primitives to Use

Integrating a motion primitive library in a multi-modal planner is straightforward. Consider INCREMENTAL-MMPRM. Suppose transition-sampling search is expanding the mode graph from a configuration q at a stance σ, and an appropriate primitive is chosen. Then, rather than expanding to an adjacent stance σ' by selecting an arbitrary new contacts, it uses the primitive to find a contacts. Then, rather than sampling an arbitrary transition configuration q', it uses the primitive to find a transition. Finally, if single-mode planning is needed to connect q and q', the multi-modal planner uses the primitive to find the path.

But it is not clear how to choose an appropriate motion primitive, or whether one should be used at all. Choosing the wrong primitive will waste computation time, or produce worse motion than a better choice. Aside from some obvious matches (like picking a flat ground step primitive on flat ground), the choice still remains unclear. Testing dozens or hundreds of motion primitives is impractical.

In preliminary experiments, a learning-based approach seems promising. We estimate a statistical model of primitive behavior (computation time, success rate, quality) as a function of local terrain features. This helps choose a primitive that is likely to produce a high-quality motion quickly. Future work might aim to combine this approach with good global heuristics learned directly from the primitive library.

4.2 Utility-Centered Expansion for a Pushing Task

Multi-modal planning algorithms can be applied to motion planning for the ASIMO humanoid robot to push an object on a table (Figure 4.10). Pushing is a potentially useful form of manipulation when an object is too large or heavy to be grasped. This section presents a technique to speed up the planner.

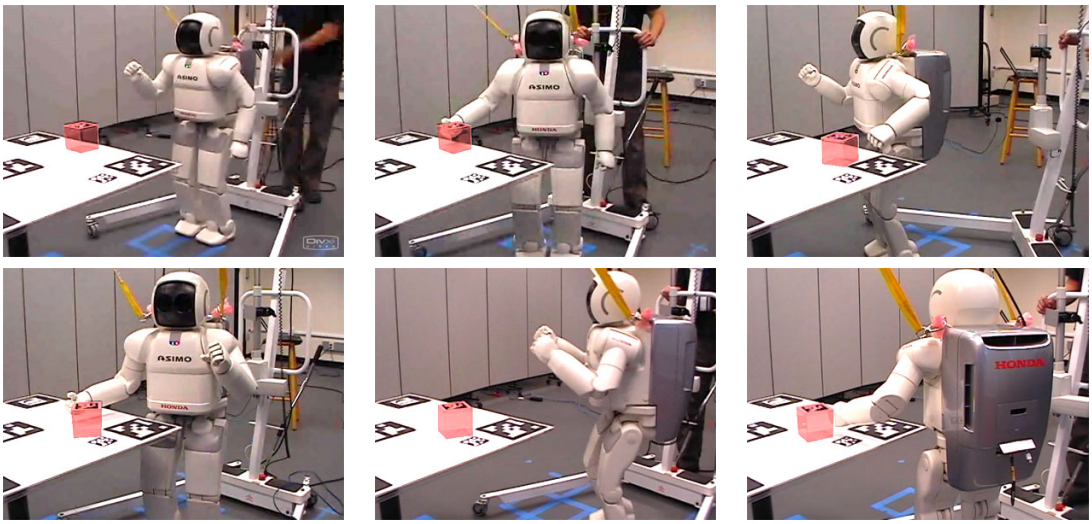


Figure 4.10: Pushing a block with the ASIMO robot. The block is shaded red for clarity.

This task combines legged locomotion and manipulation, and the multi-modal planning algorithms from Chapter 2 directly apply. But an additional constraint is imposed: while ASIMO walks, the swaying of its body prohibits accurate hand positioning, so ASIMO must stand still while reaching for and pushing the object. If we choose where ASIMO stands arbitrarily, then to move the object even a moderate distance, the robot makes many small pushes, repositioning itself each time. This wastes time and looks unnatural.

Because the reachable workspace of ASIMO's hands is fairly limited, the planner must position the robot carefully to execute a single long-distance push. To help choose such locations, we precompute tables of push *utility*, the expected distance the object can be pushed. Then, the planner uses a mode-sampling strategy that first picks a high-utility push as a subgoal, then plans "backward" to find a sequence of mode switches to execute the push. The technique enables the planner to solve easy problems in less than a minute, and difficult problems with obstacles in minutes on a PC. This work, and the multi-modal planner in which it was integrated, was presented in [59].

4.2.1 Assumptions and System Modeling

We plan for ASIMO to push an object across a horizontal table, moving one arm at a time for convenience. Given a desired translation and/or rotation for the object, the planner computes a path for the robot to follow, and an expected path for the object.

The planner assumes a perfect geometric model of the robot, the object, and all obstacles. Other physical parameters are specified, e.g. the object's mass and the hand-object friction coefficient. The robot is not permitted to collide with itself or obstacles, and may only touch the object with its hands. It must obey kinematic limits. The object may not collide with obstacles or fall off the table. We also require that the object be visible from the robot's cameras while pushing to avoid some unnatural motions (e.g. behind-the-back pushes).

We assume the object moves quasi-statically (slides without toppling and comes to rest immediately), and can be pushed without affecting the robot's balance. To predict the object's motion when pushed, we restrict ourselves to use stable pushes [93]. This imposes additional constraints on the hand and object motion during a push.

A configuration q combines a robot configuration q_{robot} and an object configuration q_{obj}. ASIMO's walking subsystem allows fully controllable motion in the plane, so leg joint angles can be ignored. Thus, q_{robot} consists of a planar transformation $(x_{robot}, y_{robot}, \theta_{robot})$, five joint angles for each arm, and a degree of freedom for each hand ranging from open to closed. Since the object slides on the table, q_{obj} is a planar transformation $(x_{obj}, y_{obj}, \theta_{obj})$. In all, the configuration space $\mathcal{Q}$ is 18 dimensional.

4.2.2 Walk, Reach, and Push Modes

The motion of the robot/object system is divided into five mode families: walking, reach left, reach right, push right, and push left. Each mode has its own motion constraints, specified as follows. In walk modes, only the base of the robot $(x_{robot}, y_{robot}, \theta_{robot})$ moves. The arms must be raised to a "home configuration" that avoids colliding with the table while walking. In reach modes, only a single arm and its hand may move. In push modes, the hand is in contact with the object. The

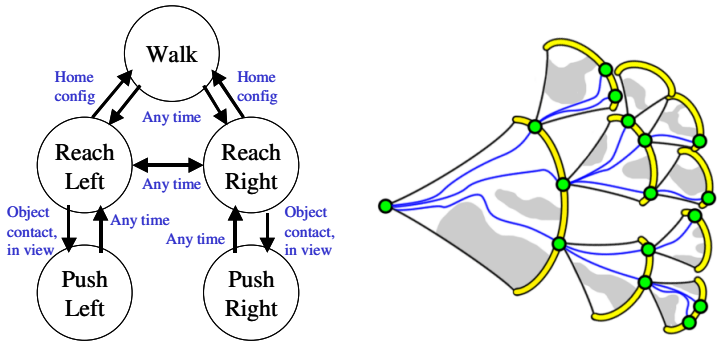


Figure 4.11: (a) Diagram of permitted mode transitions. (b) Growing a multi-modal planning tree. Each "fan" depicts a mode's configuration space, with transitions in yellow. Green nodes are feasible configurations, connected with feasible single-mode paths in blue.

object moves in response to the arm motion according to push dynamics, and reciprocally, the dynamics impose constraints on arm motions (Sect. 2.4). Additionally, the object must lie in the robot's field of view.

The following mode transitions are permitted (Figure 4.11a). Walk-to-reach, reach-to-reach, and push-to-reach are allowed from any feasible configuration. Either the left or right arm may be chosen. Reach-to-walk is allowed if the arms are returned to the home configuration. Reach-to-push is allowed when the hand of the moving arm contacts the object.

We restrict the planner to use pushes that, under basic assumptions, rotate the object predictably. These stable pushes must be applied with at least two simultaneous collinear contacts, such as flat areas on the robot's hand. Given known center of friction, surface friction, and contact points, one can calculate simple conditions on the stable centers of rotation (CORs) [93]. The planner uses these conditions to select stable CORs c such that rotating the hand in the plane about c will predictably rotate the object about c as well. Pure translations are represented by a COR at infinity.

4.2.3 Expansion Strategies

The multi-modal planner of [59] builds a tree T of configurations, rooted at the start configuration, and connected with feasible single-mode paths (Figure 4.11b). To grow T, it repeatedly chooses an existing configuration q at mode σ, then executes an expansion strategy.

We will compare three variants of the expansion strategy. The latter two use precomputed *utility tables*, which will be described below.

- *Blind.* Picks a mode adjacent to σ at random, and plans a single-mode path to it.
- *Utility-based importance sampling.* Same as the blind strategy, but samples contacts for reach-to-push transitions according to expected utility.
- *Utility-centered.* Uses the utility tables to select a high-utility push that moves q_{obj} toward a target configuration q_{des}, and a sequence of modes to achieve that push.

Utility-centered exploration has an additional parameter q_{des}, which can help heuristics guide the distribution of configurations in T, for example, toward the goal or sparsely sampled regions in configuration space. Such heuristics are not the focus of this section; details can be found in [59].

Blind Expansion

Given a configuration q at mode σ, blind expansion samples a transition configuration q' at an adjacent mode σ', and if q' is feasible, plans a single-mode path y to connect q and q' as usual. We first choose an adjacent mode family, then sample q' to achieve that type of transition. We sample q' as follows:

- *Walk-to-reach.* Sample a body position between a minimum and maximum distance from the object and a body orientation such that the object lies within the robot's field of view.
- *Reach-to-walk.* Move the arm to the home configuration.

- *Reach-to-reach.* Use any configuration.
- *Reach-to-push.* Let S_{hand} and S_{obj} be the surfaces of the hand and the object. We predefine a number of contact points $C_{cand} \subset S_{hand}$ that are candidates for stable pushes. Sample a point p_{hand} from C_{cand} and a point p_{obj} from S_{obj}, with normals n_{hand} and n_{obj}. Starting from a random arm configuration, use numerical IK to simultaneously place p_{hand} at p_{obj} and orient n_{hand} to $-n_{obj}$.
- *Push-to-reach.* A push-to-reach transition can be taken at any time. So we choose a transition implicitly (see Section 2.2.1) by planning a single-mode path y, and set q' to the endpoint of y.

After q' is picked, a single-mode path is planned using standard PRM techniques.

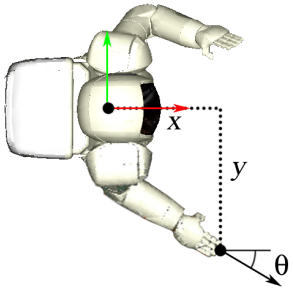
Utility computation

In reach-to-push sampling, only a small portion of S_{obj} is reachable from a given point on the hand. For each p in C_{cand}, we precompute one table that identifies the reachable region R on S_{obj}, and another table that estimates the utility of points in R.

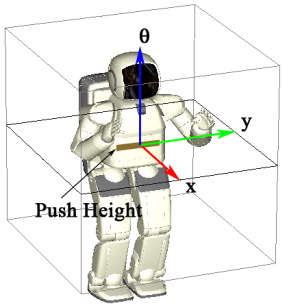
When pushing, the normal n at p must be horizontal in world space. We fix a height of pushing h, constraining the vertical coordinate of p. This define a 3D workspace $\mathcal{W}$ of points (x, y, θ), where (x, y) are the horizontal coordinates of p and θ is the orientation of n, relative to the robot's frame (Figure 4.12a). We precompute two grids, `Reachable` and `Utility`, over $\mathcal{W}$ as follows.

`Reachable` stores 1 if the contact is reachable and 0 otherwise (Figure 4.12). We initialize `Reachable` to 0, and then sample the 5D space of the arm joints in a grid. Starting at each sampled configuration, we run IK to bring the height of p to h and reorient n to be horizontal. If successful, we check if the arm avoids collision with the body and the point p is in the robot's field of view. If so, we mark `Reachable`$[(x, y, \theta)]$ with 1, where (x, y, θ) are the workspace coordinates of p and $[\cdot]$ denotes grid indexing.

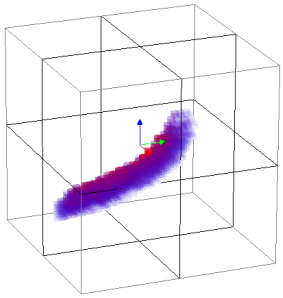
`Utility` stores the expected distance the contact can be pushed in the absence of obstacles, calculated by Monte Carlo integration through `Reachable` (Figure 4.12).



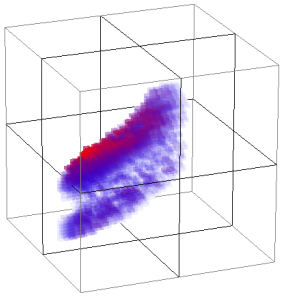
(a) Workspace coordinates of the right fingers.



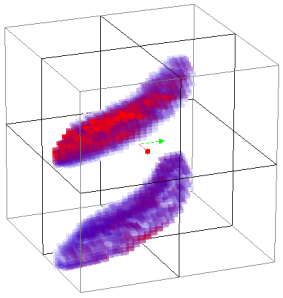
(b) Reference frame F.



(c) Utility table for the right fingers.



(d) Utility table for the right knuckles.



(e) Utility table for the right palm.

Figure 4.12: Utility tables (d–f) are drawn in frame F (c). Reachable cells are drawn with utility increasing from blue to red.

In $\mathcal{W}$, a push traces out a helix that rotates around some COR. We assume a prior probability P over stable CORs for a reasonable range of physical parameters of the object. Starting from $w_0 = (x, y, \theta)$ we generate a path $w_0, w1, \ldots, w_K$. For all k, w_{k+1} is computed by rotating w_k a short distance along some COR sampled from P. The sequence terminates when `Reachable`$[w_{K+1}]$ becomes 0. After generating N paths, we record the average length in `Utility`$[(x, y, \theta)]$.

Given robot and object positions, contacts along S_{obj} at height h form a set of curves B in $\mathcal{W}$. Intersecting B with the marked cells of `Reachable` gives the set of reachable object edges R.

When choosing a reach-to-push transition, utility-based importance sampling picks contacts from R with probability proportional to `Utility`. If R is empty, it does not attempt to make the transition.

Utility-centered expansion

Utility-centered expansion explicitly chooses a body position to execute a high-utility push. Given a start configuration q at stance σ, and a target configuration q_{des}, this strategy 1) chooses a robot's body and arm configuration and a high-utility push for a reach-to-push transition q_{push}, 2) plans a sequence of three modes backwards from q_{push} to q (which requires no search), and 3) plans a push path forward from q_{push}.

We elaborate on step 1. Let q_{obj} be the object configuration in the desired configuration q_{des}. Choose a point p_{obj} on S_{obj} (at height h and normal n_{obj}) and a stable push such that the object will be pushed toward q_{obj}. Next, sample a contact p_{hand} from C_{cand} and a workspace coordinate $(x_{hand}, y_{hand}, \theta_{hand})$ with probability proportional to `Utility`. Then, compute the body coordinates that transform (x_{hand}, y_{hand}) to p_{obj} and rotate θ_{hand} to θ_{obj}, where θ_{obj} is the orientation of $-n_{obj}$. Repeat until the body position is collision free. Fixing the body, sample the arm configuration of q_{push}, using IK to position phand to p_{obj} and orient n_{hand} to $-n_{obj}$.

This strategy places subgoals directly in reach-to-push transitions, which can be considered "bottlenecks" because of their low success rate. This is an example of a *most-constrained-first* subgoal selection strategy, which will be examined in more detail in Section 5.8.

Strategy	Walk-to-reach	Reach-to-push	Pushes > 1cm	Overall
Blind	0.63	0.20	0.29	0.037
Utility importance	0.59	0.33	0.79	0.16
Utility-centered	1	0.47	0.66	0.31

Table 4.1: Fraction of feasible walk-to-reach transitions, reach-to-push transitions, acceptable pushes, and walk-reach-push cycles.

Strategy	Modes / push	Time / push (s)	Average push (cm)	Push rate (m/s)	Tgt. seek rate (m/s)
Blind	10.0	0.956	6.7	0.070	0.0302
Utility importance	5.99	**0.325**	8.2	0.254	0.111
Utility-centered	**5.08**	0.404	**13.3**	**0.329**	**0.257**

Table 4.2: Expansion strategy timing experiments. Bold indicates best in column.

Experimental Comparison

First, we measure the fraction of feasible mode transitions attempted by each strategy. We ran the three strategies starting from a configuration similar to the first frame of Figure 4.10. Each strategy is initialized with a walk mode, and terminates after a single push has been executed (requiring three transitions, from walk to reach to push). We reject any "insignificant" pushes, defined as moving the object less than 1cm. The results, averaged over 1,000 runs, are shown in Table 4.2.3. The blind strategy produces less than half as many significant pushes than the other strategies. However, reach-to-push transitions are still a bottleneck for utility-based importance sampling.

Next, Table 4.2.3 measures the expansion time and rate, paying particular attention to the distance the object is pushed per unit of planning time. The first column measures the number of modes explored before terminating, and the second measures the time elapsed. The third and fourth columns measure the distance of the terminal push, and the distance divided by computation time. The final column measures the distance (if positive) the object was pushed toward a target position q_{des}. Here, q_{des} was picked at random.

These results verify that picking contacts blindly is rarely successful, and the

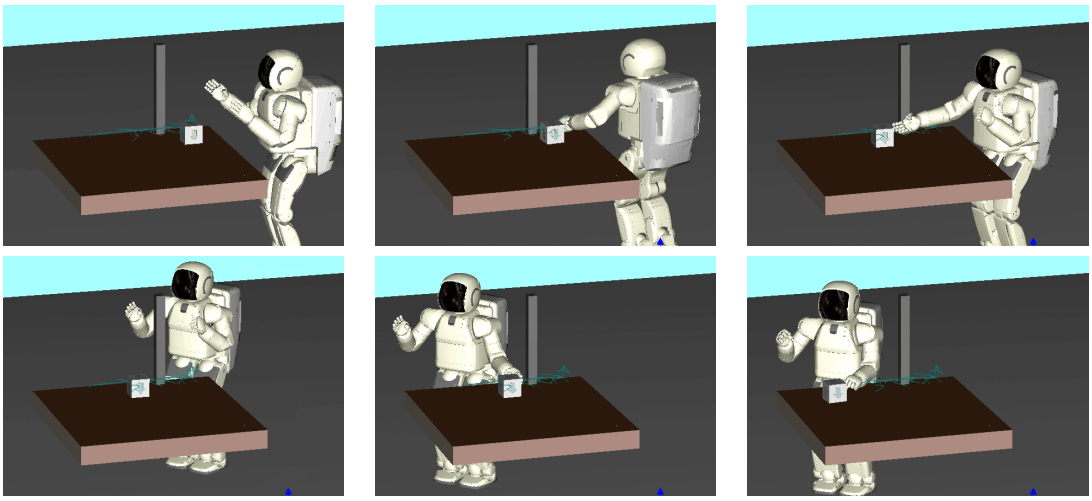


Figure 4.13: Pushing a block while avoiding a vertical obstacle.

utility table precomputation is valuable for selecting contacts. Moreover, utility-centered expansion almost doubles the average distance per push. These observations, coupled with its useful ability to guide planning toward a desired configuration, makes utility-centered expansion the preferred expansion strategy.

4.2.4 Experiments with Multiple Pushes

The utility-centered strategy was implemented in a tree-growing planner, with various heuristics to choose the target q_{des} [59]. Figure 4.13 shows the planner's solution to a moderately difficult problem, where the robot needs to carefully choose where and how it pushes the object to avoid colliding with the vertical obstacle. The relatively long-distance push chosen in frame 3 enables ASIMO to continue pushing the object in frame 5. This trajectory was found in about four minutes on a 2GHz PC. Figure 4.10 shows a sequence of pushes executed on the physical ASIMO, which was planned in about one minute on a similar PC.

The robot-relative pose of the object and table were visually sensed once at the beginning of execution using the robot's stereo cameras and markers of known size. The planned path was executed without additional visual feedback. Localization was performed using the robot's internal odometry. Despite errors in odometry and the

initial sensing, ASIMO executes several pushes successfully. In our experiments, 4–6 pushes typically can be executed before resensing is needed.

Chapter 5

Decision Theoretic Planning

The planning subroutines in previous chapters often follow a hierarchical design pattern: divide a complex problem into simpler subproblems, which are solved individually. This pattern is seen in low-level subroutines, such as configuration sampling and feasibility testing, up to the highest-level organization of multi-modal search. When solving such problems, an execution strategy must be chosen. Which subproblems should be solved? What algorithms should be used to solve them, and with which parameters? Often, these algorithms behave stochastically (a result of randomized algorithms or a stochastic input) or unpredictably (a result of inherent complexity), so we are unable to predict the subproblem success rate, computation time, or output quality beforehand. Thus, these choices must be made under *uncertainty*.

This chapter takes a decision theoretic approach to optimize execution strategies. The key concept is to model the planner as a *decision-making agent* in a *belief space*. The planner solves subproblems using computational *tests*. Test results change the belief state. We maintain beliefs on the uncertain behavior of tests, estimating success rate, computation costs, and output quality. This defines a Markov decision process (MDP) that can be solved to minimize computational costs and maximize output quality.

I apply this framework to a wide range of subroutines used in multi-modal and uni-modal sample-based planners:

- Section 5.3 improves a "shortcutting" technique [47, 48] for smoothing PRM-generated paths.
- Section 5.4 optimizes the constraint testing order in configuration feasibility tests (Section 3.2).
- Section 5.5.3 addresses perturbation-based sampling routines used to help find transition configurations (Section 3.3.3), and in adapting motion primitives (Sections 4.1.4, 4.1.5, and 4.1.6).
- Section 5.6 addresses *fuzzy planning*: allocating computation among randomized sample-based tests. We apply these analyses to balance expansion and refinement in the INCREMENTAL-MMPRM algorithm (Section 2.4) and select transitions in search among feasible transitions (Section 2.4.2).
- Section 5.7 studies fuzzy planning in the context of single-mode planning to an endgame region. This procedure is used to switch modes in multi-modal planning (see Section 2.2.1.
- Section 5.8 studies why a bottleneck-based strategy works well for the manipulation planner of Section 4.2.

These optimized subroutines were implemented in the locomotion planner of Chapter 3, and to a lesser extent the manipulation planner of Chapter 4. Though overall performance gains are prohibitively expensive to measure directly, we estimate that the optimized planners are several times faster.

5.1 Related Work

5.1.1 High-level optimization of computer programs

Most existing techniques for high-level program optimization directly compare execution strategies by measuring overall performance. For example, genetic programming [6] generates program variants, and compares them using a fitness function.

However, in planning, the number of possible execution strategies is enormous, so many strategies will need to be compared. Furthermore, under stochastic input or randomization, evaluating performance of even a single strategy is troublesome, requiring evaluation of difficult integrals.

5.1.2 Decision Making under Uncertainty

Markov decision processes (MDPs) are a key framework for studying planning under uncertainty (see [16] for a survey). Specifically, we are operating with partially observable MDPs (POMDPs), because the variables that define the planner's belief state are not directly observed. Solving POMDPs in the general case is extremely computationally hard [105, 94]. However, some planning problems addressed in this chapter have a specific well-studied structure, for which solutions can be computed efficiently. For instance, the path shortcutting problem can be cast as an n-armed bandit problem [9]; the constraint testing problem is similar to boolean-formula testing [53]; and part of the configuration repair problem can be cast as a max n-armed bandit problem [38].

5.1.3 Uncertainty in Motion Planning

Uncertainty is fundamental to PRMs and other sample-based planners — not uncertainty in the motion model, but rather, uncertainty in beliefs about the feasible set. PRMs can be interpreted as having implicit hypotheses about the shape of the feasible set, which become more accurate as the planner probes the space with feasibility tests [63]. This suggests that PRMs might probe the space more efficiently by using better initial hypotheses, or exploiting the information contained the probing history. These ideas implicitly form the basis of tweaks to PRM sampling strategy [15, 61, 62, 66], connection strategy [3, 117], and distance metrics [3]. Another line of work aims to build explicit probabilistic models of free space [28], and chooses where to sample as if the planner were an information-gathering [29] or cost-minimizing agent [30]. An adaptive sampling strategy was implemented by treating sampler parameter selection as an n-armed bandit problem [65].

5.2 Modeling Belief Space

This section describes how to cast a planning problem as a POMDP. Consider a broad definition of "planner" as a computer program that produces its output by solving subproblems using algorithms that behave with uncertainty. I assume the program is executed sequentially, and uncertainty means the output and cost of the subproblem is either stochastic (a result of randomization or stochastic input) or unpredictable (complex enough to be considered a black box).

Belief space In the scope of a given problem, the planner operates on a number of logical *statements*. Examples are "configuration q is feasible", "q and q' can be connected", or "set A is empty". To a statement S, assign a belief p in $[0, 1]$, with $p = 0$ meaning S is invalid (certain falseness) and $p = 1$ meaning S is valid (certain truth). If S is valid or invalid, it is a *factual* statement, and is otherwise a *hypothetical* statement. An assignment of beliefs to all statements in the scope of a problem defines a *belief state*, and the set of all possible belief states is the *belief space*.

Tests The planner works by choosing a sequence of *tests*, primitive operations that modify the belief on hypothetical statements. Upon observing the results of executing a test, the planner moves to a new belief state. Some common types of tests are:

- *Exact tests.* Determine the factuality of a statement S exactly, with probability of success equal to the belief on S. For example, a test might check if a constraint is violated.
- *Partial tests.* The belief on S is modified without becoming a fact. For example, after testing that a contact lies within some reach bounds, the contact is more likely to be reachable. This can be modeled as an exact test on an auxiliary statement S', where S and S' are dependent.
- *Incremental tests.* The test can be controlled step-by-step, incrementally changing the belief on S. This class includes probabilistically complete tests like PRM planning, which are examined in detail in Section 5.6.

Tests may modify the beliefs of several statements simultaneously. For example, if statements A and B are likely to have the same truth value, and the planner demonstrates that A is true, then it becomes more likely that B is true. The effects of tests on the belief state are together called the *belief dynamics*.

Costs and Rewards The problem is considered solved when the belief state contains certain facts. For example, a feasibility-checking problem is solved when it has determined that one constraint is violated, or all constraints are satisfied. The planner terminates (or may choose to terminate), and an output is produced. The utility function consists of positive *rewards* received by the planner upon termination and negative *costs* incurred during execution. Rewards assess the output quality. Typically we define failure to have reward zero. Costs measure the cost of execution, including computation time and resource usage. The designer should weight rewards and costs reflecting his or her judgment of the relative importance of these factors. The optimal *execution strategy* (i.e. policy) terminates with the maximum expected value of the sum of rewards received minus the sum of costs incurred.

Belief modeling and dynamics. In full generality, a belief state on statements $\mathcal{S}$ and future test results $\mathcal{T}$ forms a joint probability distribution $Pr(\mathcal{S}, \mathcal{T}|Z)$. Here, Z represents *background knowledge* established prior to the current planner state, which will be discussed below. After executing a test T, the belief state should change to $Pr(\mathcal{S}, \mathcal{T}|T \text{ succeeds}, Z)$ with probability $Pr(T \text{ succeeds}|Z)$, or to $Pr(\mathcal{S}, \mathcal{T}|T \text{ fails}, Z)$ with probability $Pr(T \text{ fails}|Z)$.

Background knowledge Given a new instance of a problem, the planner must begin at an initial belief state, infer test costs, and infer terminal rewards. Furthermore, there are often too many logical statements to explicitly represent, and their costs and beliefs must be initialized on demand. We must infer these values from some other information. Call this information *background knowledge*. This knowledge persists between planner executions and is independent of problem input.

Representations A joint distribution is impractical to compute or represent explicitly because its size is exponential in the number of statements. We will often assume statement independence, in which case the joint distribution is simply the product of distributions of individual statements in $\mathcal{S}$. More refined strategies might use assumptions of conditional independence, for example, representing variables in a sparse Bayesian network. We represent background knowledge using a variety of machine learning and statistical models, e.g., logistic models, Bayesian networks, decision trees, neural networks, etc.

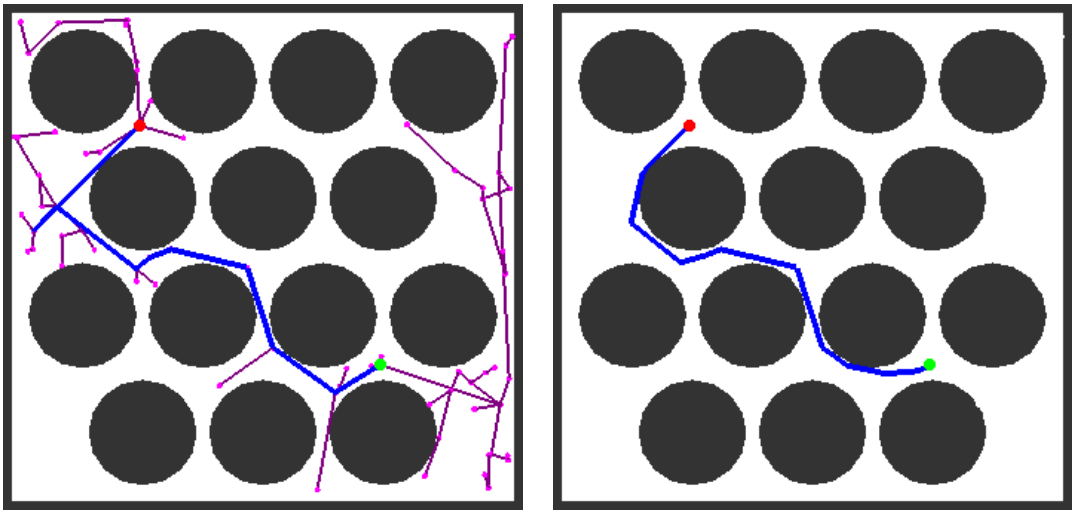
5.3 Path Shortcutting

Probabilistic roadmap planners tend to produce jerky, unnatural-looking paths due to their random exploration. A simple shortcutting method [47, 48] smoothes these paths by picking two random points A and B on a path, and testing the line segment $\overline{AB}$ for feasibility. If $\overline{AB}$ lies in free space, it replaces the portion of the path between A and B. After a handful of iterations, the largest unnecessary jerks are likely to be eliminated. However, it can take a huge number of iterations before the process converges to a smooth path. How many iterations are enough?

We formulate this as an *additive-payoff problem*, where the decision maker receives rewards that are accumulated over time. Each potential shortcut incurs a negative cost, and produces a reward only if it is successful. We show that a greedy strategy converges much faster than picking random points, and has a natural termination criterion: halt when no shortcut has positive expected utility. This strategy was implemented in path-smoothing postprocessors for both the locomotion planner of Chapter 3 and the manipulation planner of Section 4.2

5.3.1 Problem Statement

Suppose the path $y(u)$ is parameterized with u in [0,1]. Denote the length of the path between parameters u and u' as $l(u, u')$, and denote the distance between two points q and q' in C-space as $d(q, q')$. Testing the line segment between u and u' incurs cost



(a) (b)

Figure 5.1: (a) PRM planners produce jerky paths. (b) Shortcutting produces a shorter path.

$c(u, u')$. If successful, $y(u)$ is replaced with the new path, and the planner receives reward $l(u, u') - d(y(u), y(u'))$ (the amount that path length is reduced). Let $p(u, u')$ denote the estimated probability of success.

We assume $c(u, u') = c_s d(y(u), y(u'))$, where c_s is a constant reflecting the amount of computation time one is willing to spend for a unit decrease in path length.

5.3.2 Decision Theoretic Model

This is essentially a "one-pull" variant of the well-studied n-armed bandit problem [9]. Here, the decision maker can choose from n actions, $A_1, \ldots, A_n$. A stochastic payout z_k is awarded after choosing A_k, and payouts accumulate over time. If the distributions of each z_k are known, the optimal strategy is greedy and picks the choice with the highest expected reward. This is also the case for "one-pull" bandits.

In other words, the greedy choice of u and u' maximizes $p(u, u')(l(u, u') - d(q, q')) - c(u, u')$. If we ignore dependencies between candidate shortcuts (for example, when the range of two shortcuts overlap), then greedy choice is optimal.

5.3.3 Belief Estimation

The performance of the greedy strategy depends on the quality of the estimate $p(u, u')$. The probability that a line segment is feasible decreases with distance between its endpoints [113]. Also, a line segment is likely to be invalid if nearby line segments are invalid. We keep a history of infeasible configurations $\mathcal{I}$ (which may be initialized with samples from PRM planning) and updated as shortcuts fail. We assign a belief to each shortcut as a function of segment length and the distance between the segment and the closest configuration in $\mathcal{I}$.

In the following experiments, we use a crude method to estimate $p(u, u')$ given $\mathcal{I}$. We tested the feasibility of 100,000 randomly sampled line segments, and built a histogram $p_0(d)$ of success rate indexed by distance d. We then weight the baseline probability $p_0(d(q, q'))$ by a function of the distance d' to the closest infeasible configuration. The experiments below used $f(d') = 1 - e^{-\alpha d'}$, where α was chosen by a small amount of tuning. A better method might estimate the joint success rate as a function of $d(q, q')$ and d'.

5.3.4 Experimental results

We demonstrate this for a C-space with regularly spaced circular obstacles in a unit square (Figure 5.1). The shortest path between two points may contain curves on the C-space obstacle boundaries (this is also true in C-spaces for robotic mechanisms with revolute joints). A good piecewise linear approximation to the shortest path requires a huge number of line segments.

To compare performance across varying start and goal configurations, we normalize reward by dividing by the straight-line distance between the starting points. We set the cost proportionality constant c_s to 0.01. Figure 5.2 compares the randomized strategy with a greedy, adaptive strategy. 200 random start and goal locations were sampled and connected using an RRT planner [86]. Starting at the output path, random shortcuts were performed for 1000 iterations. For the same starting path, adaptive shortcuts were performed until the strategy chose to terminate. The set of infeasible configurations $\mathcal{I}$ is initially empty.

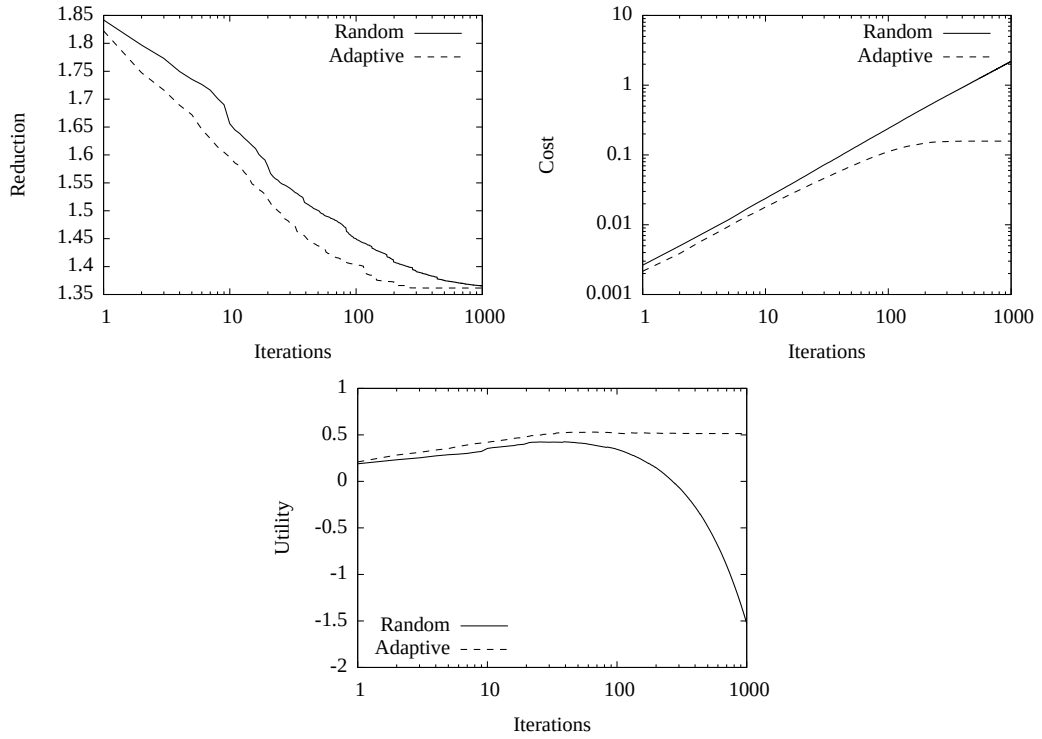


Figure 5.2: Experimental relative path length, accumulated cost, and utility. Averaged over 200 plans with random start and goal configurations. Iteration numbers are plotted on a log scale.

Initially, the adaptive strategy reduces the path length quickly. Throughout execution, it achieves a given reduction in path length about two or three times faster than the randomized strategy. As more iterations are taken, shortening the path becomes increasingly harder. The adaptive strategy terminates naturally when the cost of making a shortcut exceeds the expected reward. All adaptive runs terminated by iteration 200.

5.4 Constraint Testing

Consider the problem of testing a configuration for feasibility, where the configuration must pass a number of feasibility conditions. There are hundreds of conditions to test when a robot consists of multiple rigid links, each of which must be tested for

collision against the environment, and each pair of links must be tested for self-collision. Conditions range from rarely violated (say, collision between a foot to a head) to frequently violated (support polygon), and their tests range from extremely slow (torque limit tests) to extremely fast (contact tests). Here we show that moderate speed gains are achieved by rearranging the condition testing order such that many configurations are rejected by a small number of inexpensive tests. These techniques are implemented in the locomotion planner of Chapter 3.

5.4.1 Decision Theoretic Model

Represent each of the n feasibility conditions as statements $A_1, \ldots, A_n$. Suppose A_k is valid with probability p_k, with truth value determined exactly by a test of cost c_k. We seek to determine the truth value of $A_1 \wedge \cdots \wedge A_n$ with the minimal cost (rewards do not matter, because the truth value cannot be changed by the planner). At most, the cost will be $\sum c_k$, because we can evaluate the validity of all statements and evaluate the formula's truth value.

When statements are dependent, the optimal strategy can be expensive to compute [53]. But if all statements are independent, the optimal strategy tests the statement A_k that minimizes $c_k/(1-p_k)$ [53]. This can be explained intuitively in that if F is actually true, every hypothetical statement must be tested regardless of strategy. The optimal strategy, therefore, tries to prove F false with the least expense. This takes the form of minimizing the ratio of cost to the probability of early termination.

This may sometimes run contrary to the "lazy" intuition, because a more expensive test that many configurations fail (high c_k but low p_k) may be a better choice than a cheap test that is almost always successful (low c_k but high p_k).

5.4.2 Experiments

We compared this strategy to a link-order strategy in collision detection. Using the legged locomotion planner of Chapter 3, we gathered data on collision tests for ATHLETE and HRP-2 on a variety of terrains. The above strategy was observed to be about 15–20% faster than a link-order strategy overall. This gain is small, but

significant considering that a single motion planning query will require on the order of millions of collision test.

Hypothetically, if the feasibility conditions were independent, the optimal strategy would have been 25–30% faster than the link-order strategy. This discrepancy is caused by dependencies between constraints, such as those that arise from topologically close robot links (the foot and ankle, for example).

5.5 Configuration Repair

Consider the problem of "repairing" an infeasible configuration q_0 to find a feasible configuration q similar to q_0 (as measured by some distance metric). This process is used in seed sampling (Section 3.3.3), planning with motion primitives (Chapter 4), and is also used in free-space deformation strategies for PRM planning.

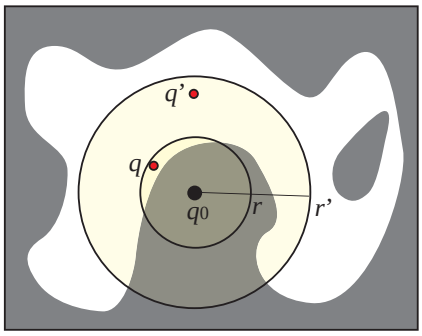
Sampling at random ignores the similarity objective. Numerically optimizing similarity is expensive. Instead, suppose we use a neighborhood sampler $\textsc{Sample}(q_0, r)$ that samples uniformly from the neighborhood of radius r around q_0. How should we select r? If r is too small, the success rate is lower, meaning we must sample many more configurations before finding a feasible one; too large, and we produce configurations that are dissimilar from q_0 (Figure 5.3a).

Using a decision theoretic analysis, we derive a strategy that finds nearby configurations with a small number of samples and feasibility tests, and improves upon other strategies that require tuning parameters (e.g., maximum number of samples, neighborhood size, neighborhood growth strategy) that have no clear relation to problem attributes.

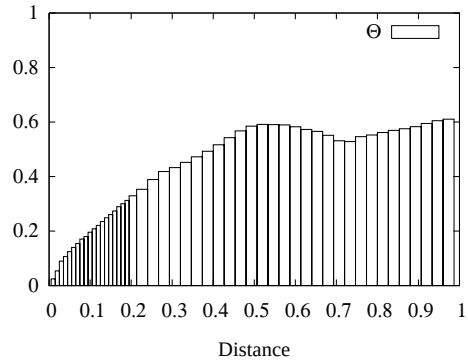
5.5.1 Problem Statement

The planner uses two operations:

- Execute $\textsc{Sample}(q_0, r)$ to produce a sample q, and add q to a set $\mathcal{S}$. This incurs cost c_s.
- Remove a sample q from $\mathcal{S}$, and test it for feasibility. This incurs cost c_f.



(a) Choosing between large and small neighborhood radii.



(b) Feasibility histogram as a function of distance, for an example problem.

Figure 5.3:

A feasible sample establishes a *payout*, with value decreasing in distance from q_0. Let the distance metric be $d(q, q')$ and let the payout $z(q) \equiv z(d(q, q_0))$ be decreasing in distance. The sampler has the option of terminating and receiving the payout as its reward, or continuing to sample in the hope of establishing a higher payout. As usual, the planner should optimize reward minus the sum of incurred costs.

5.5.2 Belief Representations

We maintain estimates of the probability that a sample q is feasible, as a function of its distance r from q_0. We divide the radius into k buckets, with $r_{i-1} \leq r < r_i$ for $i = 1$ to k, with $r_0 = 0$ and $r_k = \infty$. Each bucket defines a hollow ball around q_0. For each bucket $[r_{i-1}, r_i)$, the fraction of the hollow ball contained in the feasible set $\mathcal{F}$ is some fraction θ_i. Denote Θ as the histogram of all θ_i (Figure 5.3b).

Our experiments will compare two types of background knowledge used to define Θ:

1. *Independent beliefs* We assume all θ_i are independent and follow beta priors. We estimate Θ by sampling 100,000 pairs of points q_1 and q_2. For each infeasible q_1, we recorded the feasibility of q_2 as a function of distance $d(q_1, q_2)$ in the histogram.

2. *Classified beliefs* (denoted *Class. Beliefs*). We assume all θ_i are independent given a *class* variable C. We allow C to take 8 values. We estimate a parameter vector Θ_c for each class c as follows. We sampled 10,000 infeasible points. For each point q_k, we built a local histogram of feasibility Θ_k around q_k using 1,000 samples. We then find class histograms Θ_c and class labels c_k for each of the q_k by maximizing the likelihood that each Θ_k is generated by Θ_c. These are optimized using expectation maximization.

   During configuration repair, each class is set to have uniform prior. The prior acts as weights on Θ_c from which we infer the overall θ_i. When samples are found feasible or infeasible, we update the class distribution using Bayes rule and infer the updated θ_i.

For all histograms, we defined 50 evenly spaced buckets between 0 and 1.

5.5.3 Choosing a Sample to Test: Maximizing Payout

Suppose $\mathcal{S}$ has been populated by a number of samples. If we were to test a sample for feasibility, which one should be picked? This is a general problem where we wish to find a valid statement (e.g., a feasible configuration) that maximizes some payout (e.g., a similarity measure). This *maximum-payout model* is similar to additive-payout models, except only the *maximum* payout is reward upon termination. Similar models have attracted some recent attention for modeling parameter and algorithm selection problems [38].

In the independent belief case, the optimal strategy is quite simple: it picks the statement that maximizes a simple *score*. For all q, define the score $s(q) = z(q) - c_f/\theta_i$, where $r_{i-1} \leq d(q, q_0) < r_i$. Let Z^T be the highest established payout by step T. On step T, the strategy tests the q in $\mathcal{S}$ that maximize $s(q)$, or halts if $s(q) \leq Z^{(T)}$ (in this latter case, the expected one-step utility is no greater than halting). This result can be proven using induction on n if all samples in $\mathcal{S}$ are in separate buckets (and are therefore independent). If multiple samples lie in a single bucket, the proof is a bit more complex, but the result still holds. Also note the max-score strategy halts on the first feasible test, so we can discard any samples in $\mathcal{S}$ that are not immediately

tested.

The max-score strategy might not be optimal for the classified belief case, because the class variable causes all configurations in $\mathcal{S}$ to be dependent. However, it still works extremely well in practice.

5.5.4 Selecting a Sampling Radius

Adding the possibility of sampling new configurations complicates matters. If sampling in a neighborhood of radius r is the optimal choice, then after SAMPLE(q_0, r) produces a new configuration q, the next optimal choice either tests the feasibility of q, or samples again. If q is infeasible, the next optimal choice must sample again (because we are in the same position in which we started). Therefore, we should analyze the conditions under which sampling halts.

Let the halting criterion be that a configuration q has been sampled with $s(q) \geq \zeta$. It remains to choose r and ζ to yield the optimal policy. By this halting rule, the expected utility of sampling is

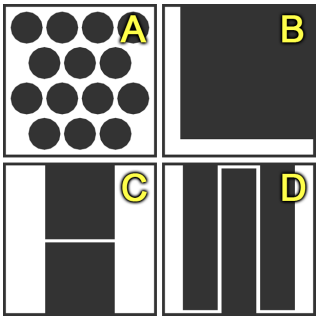
$$\begin{aligned} U_s(z) = &-c_s + Pr(s(q) < \zeta)U_s(z) - Pr(s(q) \geq \zeta)c_f \\ &+ \int_q I[F_q \text{ and } s(q) \geq \zeta]U_s(z(q))dq + Pr(\neg F_q \text{ and } s(q) \geq \zeta)U_s(z) \end{aligned} \tag{5.1}$$

where the utility depends on the current established payout z, q is the configuration produced by SAMPLE(q_0, r), and F_q denotes the event that q is feasible. Rearranging, we get

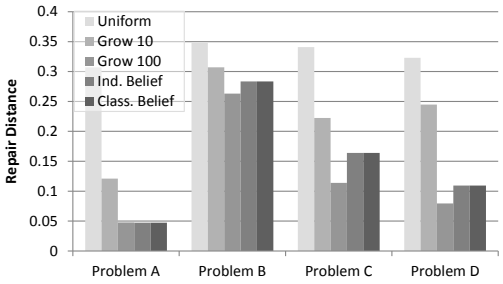
$$Pr(F_q \text{ and } s(q) \geq \zeta)U_s(z) = -c_s - Pr(s(q) \geq \zeta)c_f + \int_q I[F_q \text{ and } s(q) \geq \zeta]U_s(z(q))dq. \tag{5.2}$$

When $z(q)$ is the established payoff, $U_s(z(q)) \geq z(q)$ because it could immediately halt. We approximate $U_s(z(q))$ with $z(q)$. This means that we are optimizing over strategies that halt on the first feasible sample, a decision justified by the observation in the previous section.

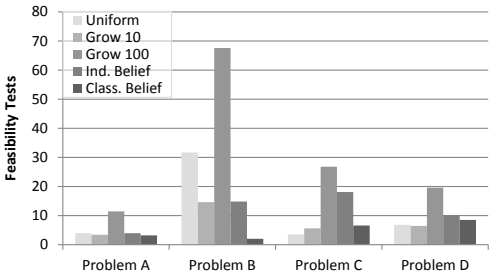
Then we discretize by buckets. Let $r = r_m$ for some m. Assuming that $z(q)$ is



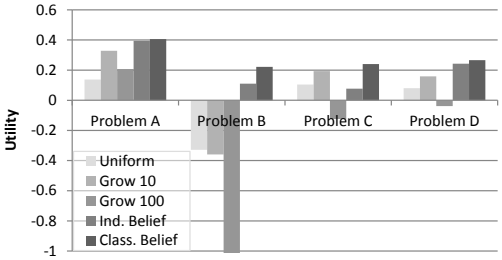
(a) Test configuration spaces.



(b) Distance between q_0 and output q.



(c) Number of feasibility tests.



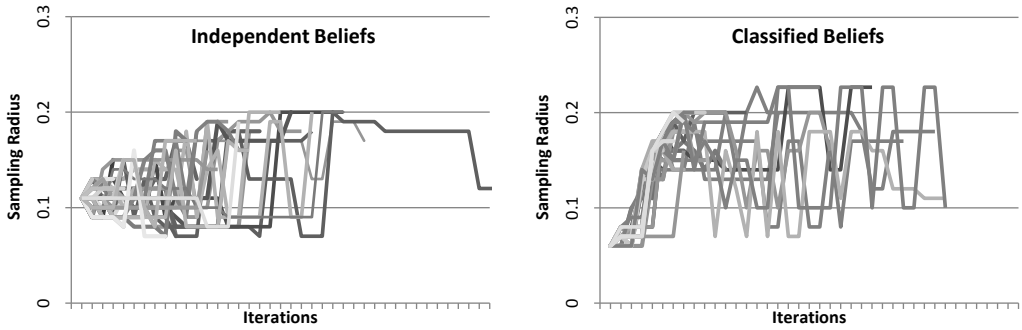
(d) Utility.

Figure 5.4: Experiments on configuration repair strategies, averaged over 1000 runs.

constant over each bucket, we get

$$\begin{aligned} U_s(z) \sum_{i=1}^{m} I[z_i - c_f/\theta_i \geq \zeta] \frac{v_i}{V_m} \theta_i = \\ -c_s - c_f \sum_{i=1}^{m} I[z_i - c_f/\theta_i \geq \zeta] \frac{v_i}{V_m} + \sum_{i=1}^{m} I[z_i - c_f/\theta_i \geq \zeta] \frac{v_i}{V_m} \theta_i z_i \end{aligned} \tag{5.3}$$

where v_i is the volume of a bucket in C-space, V_m is the volume of the neighborhood of radius r_m, and z_i is the average payout over a bucket. Then, we use a discrete search to choose ζ and m to maximize $U_s(z)$ (we only need consider ζ that fall between buckets).



(a) (b)

Figure 5.5: Radius selection schedules for 50 runs on problem C using the two belief representations.

5.5.5 Experimental Results

We tested several repair strategies on various configuration spaces in a unit square (Figure 5.4a). Figure 5.4 reports results averaged over 1,000 randomly sampled, infeasible starting configurations. We tested three basic strategies: 1) fixing $r = 0.5$ (denoted *Uniform*), 2) growing r to 0.5 in 10 increments (denoted *Grow 10*), and 3) growing r to 0.5 in 100 increments (denoted *Grow 100*). These were run for up to 100 iterations, or until they found a feasible sample. We also tested the decision-theoretic strategy, using either the independent belief (denoted *Ind. Beliefs*) or the classified belief (denoted *Class. Beliefs*) representations. We used the reward function $z(q) = 0.5 - d(q, q_0)$ and costs $c_s = 0.005$ and $c_f = 0.01$. Background knowledge was inferred separately for each problem.

Grow 100 is likely to find a nearby feasible configuration (Figure 5.4b), but it requires a large number of samples to do so (Figure 5.4c). Grow 10 actually has a much higher average utility (Figure 5.4d). The independent belief strategy performs significantly better than the basic strategies on almost all problems.

However, it performs somewhat poorly on problem C. Here, there are roughly two types of infeasible points: those near the feasible space boundary, and those deep inside the infeasible set. If a few nearby samples are infeasible, then this indicates that q_0 is in deep. Figure 5.5 illustrates that the classified belief strategy acts appropriately

and expands the sampling radius quickly, but the independent belief strategy does not. The classified belief strategy is also fairly robust to belief estimation errors. For example, if we use the beliefs generated for problem C in Figure 5.4a on other problems, this strategy performs better than the basic strategies, and almost as well as the problem-specific beliefs.

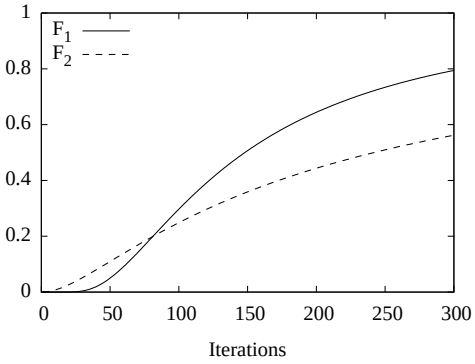
These experiments demonstrate that even with crude independence assumptions, decision theoretic strategies generally perform well. More sophisticated assumptions may be warranted to refine performance further.

5.6 Fuzzy Planning

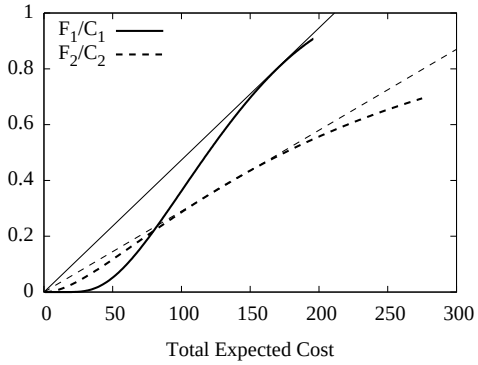
Several subproblems in multi-modal planning are solved using a probabilistically complete algorithm. Often, we must find (and solve) a feasible subproblem out of a larger set $A_1, \ldots, A_n$, some of which are infeasible. With a probabilistically complete test, the planner may fail to terminate if it attempts to test a single subproblem to completion. In Section 2.2.3, we identified that establishing a cutoff involves a difficult parameter choice and is not guaranteed to find a solution. Interleaved allocation is better, but how should computation be distributed among subproblems?

We show that the optimal policy can be determined from a *success rate profile*, an estimate of the probability that a subproblem succeeds as a function of the number of iterations. This profile can be computed easily from experiments. We apply these results to SAFT (Section 2.4.2), a search procedure that incrementally samples transition configurations. With no prior information about the feasibility of transitions, interleaving samples is optimal. However, much better strategies might use prior information to distinguish feasible transitions from infeasible ones. This work is applied to locomotion planning for a climbing robot.

We also address the exploration and refinement trade-off in the IncrementalMMPRM algorithm of Section 2.4. Here, proving an optimal strategy is difficult, but we use the intuition from previous sections to choose parameters that work well in practice when implemented in the locomotion planner of Chapter 3.



(a) A choice between two profiles.



(b) The optimal choice maximizes the ratio of probability to total expected cost.

Figure 5.6:

5.6.1 Success Rate Profiles

Assume each subproblem is independent. Let the *success rate profile* $F_k(n)$ measure the cumulative probability that A_k has succeeded by the end of iteration n (with $n \geq 0$). F_k has the property that $\lim_{n\to\infty} F_k(n) = Pr(A_k \text{ has a solution})$. Also, if the test is a coin flip with probability a, then $F_k(n) = 1 - (1-a)^n$.

Consider the simplified problem of trying to prove any statement true as quickly as possible. The optimal strategy picks A_k and n to make the greatest improvement in success per cost expenditure. After n tests are executed, the probability that A_k has been proven true is $F_k(n)$ (Figure 5.6a). Let $C_k(n)$ be the expected cost of these n tests (terminating if A_k is proven valid). If each iteration has constant cost c_k, this takes the form

$$C_k(n) = c_k \sum_{j=0}^{n-1} (1 - F_k(j)).$$

Then, we should pick k and n to maximize

$$\frac{F_k(n)}{C_k(n)}. \tag{5.4}$$

This can be interpreted visually as maximizing the slope of the graph of success rate to expected cost (Figure 5.6b). The optimal choice of A_k and n can be determined

quickly using discrete search techniques (e.g., branch-and-bound).

Once n tests are executed without success, the strategy repeats with a modified F_k. Given that n tests failed, the probability that a success is encountered in m additional tests is a new profile $\hat{F}_k(n) = (F_k(n+m) - F_k(n))/(1 - F_k(n))$. This can be interpreted visually by shifting the function F_k to have origin at $(n, F_k(n))$, and then scaling it vertically by $1/(1 - F_k(n))$.

5.6.2 Search Among Feasible Transitions

In Section 2.4.2 we presented a fuzzy search algorithm for SAFT, without precisely specifying the ordering in which samples were drawn. If each transition has equal cost, and we have no prior information about transition feasibility, then the optimal strategy is a round-robin allocation with one sample per transition. The probability p_T that a sample from transition T succeeds depends on the volume of the feasible set. The volume is a random variable with some prior distribution (e.g., a Beta prior). Therefore, the expected value of p_T decreases in the number of failed samples (i.e., each sample has *diminishing returns*). Therefore, the success rate profile decreases at a decreasing rate, and (5.4) is maximized on the first sample.

But with feasibility information, the search can be much faster. In work presented in [56], we used machine learning techniques to estimate the success curves p_T in a locomotion planner for the LEMUR robot [23]. In a lengthy precomputation stage, we generated several thousand example transitions and tested their feasibility by extensive sampling. We then trained a neural network classifier to predict feasibility as a function of several stance features. During planning, the classifiers were used to predict the feasibility of new transitions. But rather than pruning out transitions that were classified as infeasible, we used the classification error (about 23%, as measured by cross validation) to estimate the initial p_T. Compared to a uniform p_T, using the classified estimate improved the speed of SAFT (and consequently, the overall speed of the planner) by up to an order of magnitude (Figure 5.7).

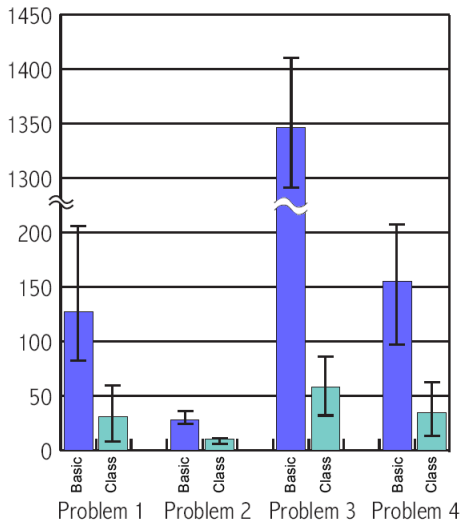


Figure 5.7: Planning time on four problems on LEMUR, with and without step feasibility classifiers. Columns plot the average over 10 runs, and whiskers plot the high and low.

5.6.3 Balancing Expansion and Refinement

The INCREMENTAL-MMPRM multi-modal planning algorithm of Section 2.4 alternates between expanding a set of candidate modes, and refining roadmaps within those modes. We use a fuzzy planning model to select between these options.

Rate-Optimal Refinement Strategy

It is difficult to compute the optimal refinement strategy for a set of candidate modes with arbitrary connectivity. To simplify, we approximate each candidate mode set by a number of disjoint paths. Specifically, we assume every expansion step produces a sequence of modes disjoint from the current candidate mode set. Then, the goal condition for refinement can be written as a nested OR/AND boolean formula.

Let this formula be $F = B_1 \vee \cdots \vee B_M$, where $B_i = A_{i1} \wedge \cdots \wedge A_{iN}$ (N need not be uniform over all B_i). Each B_i represents a disjoint path, and each A_{ij} represents a mode along the path. If tests were exact, the optimal strategy for testing F is given by computing the optimal strategies for each B_i, then picking the minimizer of $C(B_i)/P(B_i)$ (here, $C(B_i)$ represents the expected cost of testing B_i under the

optimal strategy) [53]. We use a similar strategy for the probabilistically complete case. We pick the clause B_i that minimizes $D_\pi(B_i)/G_\pi(B_i)$, where D_π and G_π are the expected cost and success probability of testing B_i under some strategy π. We allow π to be terminated after some number of iterations to ensure finite cost.

A reasonable *rate-optimal* strategy for testing B_i minimizes $D_\pi(B_i)/G_\pi(B_i)$. This strategy is given with the following theorem. Here, let the success rate profile of A_{ij} be the function $F_{ij}(n)$ as a function of iterations n, with expected cost function $C_{ij}(n)$.

Theorem 5.6.1. *The rate-optimal strategy for testing B_i chooses to refine A_{ij} with n iterations, where n is picked to minimize $C_{ij}(n)/F_{ij}(n)$ (i.e. maximizing (5.4)) and A_{ij} is picked to minimize $C_{ij}(n)/(1 - F_{ij}(n))$.*

Proof. Obviously, if $B_i = A_{i1}$ is a one-statement clause, then the rate-optimal strategy picks an iteration count n to maximize (5.4).

Now suppose $B_i = A_{i1} \wedge A_{i2}$ is a two-statement clause. Minimizing $D_\pi(B_i)/G_\pi(B_i)$ requires choosing an order of operations. First, suppose A_{i1} is tested first for some number of iterations n_1. It can be proven that if this initial test fails, the rate-optimal strategy will not test A_{i2}. Then, A_{i2} will only be tested if A_{i1} is true. The resulting clause is a one-statement clause, and the rate-optimal strategy picks the iteration count n_2 by maximizing (5.4). Therefore,

$$\frac{D_\pi(B_i)}{G_\pi(B_i)} = \frac{C_{i1}(n_1) + F_{i1}(n_1)C_{i2}(n_2)}{F_{i1}(n_1)F_{i2}(n_2)} = \frac{C_{i1}(n_1)}{F_{i1}(n_1)F_{i2}(n_2)} + \frac{C_{i2}(n_2)}{F_{i2}n_2} \tag{5.5}$$

Since n_2 is fixed, to minimize the first term, we must pick n_1 to minimize $C_1(n_1)/F_1(n_1)$ — the same as maximizing (5.4). By symmetry, if A_2 is picked first, the iteration counts n_1 and n_2 remain the same. It therefore suffices to compare the two orderings for optimality. After a bit of algebra, the rate-optimal strategy picks A_1 first if $C_1(n_1)/(1 - F_1(n_1)) < C_2(n_2)/(1 - F_2(n_2))$, and A_2 otherwise.

This result can be extended inductively to show the desired result when B_i consists of any number of statements. □

Comparing Expansion and Refinement Rates

Let B_1 be a sequence of candidate modes in the current candidate mode set. The optimal refinement rate $D_1(B_1)/G_1(B_1)$ can be computed using the above strategy. If expansion produces a sequence of candidate modes B_2, then the expansion rate is

$$\frac{c_{exp} + D_2(B_2)}{G_2(B_2)}$$

where c_{exp} is the expansion cost, and D_2 and G_2 are the expected cost and success rate of refining B_2. The strategy picks refinement or expansion by the minimum of these two ratios.

Implementation

It is fairly difficult to make accurate estimates of c_{exp}, D_2, G_2, because they require estimating the planner behavior on parts of the mode graph that have not yet been instantiated. Using a heuristic estimate of the number of modes n in the shortest path to reach the goal, and the probability profiles gathered for single-mode PRM planning, we compute estimates of D_2 and G_2. We estimate c_{exp} during expansion by measuring the rate of progress towards the goal as a function of time, and measuring the distance to the goal from the configurations at the search frontier.

This strategy was implemented in the locomotion planner of Chapter 3. We have not yet compared its performance relative to other strategies, because extensive testing is prohibitively expensive. But by inspection over a handful of experiments, the choices it makes seem intuitively correct. For example, when expansion only progresses slowly toward the goal, the planner spends more time refining. Also, if the current set of modes contains two sequences of modes leading to the goal, the planner refines the shorter one first. If the roadmaps along a sequence of modes are highly connected (indicating that a solution will likely be imminently found), then the planner spends more time refining this sequence before switching to another.

5.7 Single-Mode Planning to an Endgame Region

Suppose we are using a tree-based sample-based planner to connect a start configuration q_s to an endgame region $\mathcal{G}$. Should we plan forward from q_s until $\mathcal{G}$ is reached, or generate a goal configuration in $\mathcal{G}$, and then plan bidirectionally? Is that extra cost of generating a goal configuration worth the effort?

This problem is encountered every time a multi-modal planner attempts a mode switch. Earlier in Section 2.2.1, I argued that explicit mode transitioning is better for small endgame regions. I show that when the endgame volume is small, picking goal configurations and bidirectional planning works well because it quickly expands the "effective" endgame volume. The analysis here was implemented in the connectivity-based refinement strategy of Section 2.4.3.

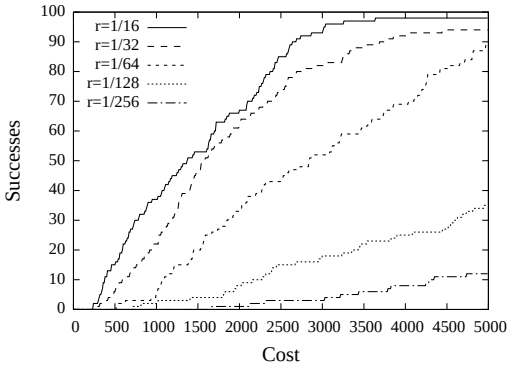
5.7.1 Benefits of Endgame Expansion

Small endgame regions make tree-growing sample-based planners converge slowly. As an example, Figure 5.8a plots the experimental success rate of the SBL algorithm on problem D of Figure 5.4a for goal neighborhoods of varying radius. This shows that the convergence rate decreases rapidly as the volume of $\mathcal{G}$ decreases. If our planner is given a problem with a small endgame region, growing the endgame would be prudent.

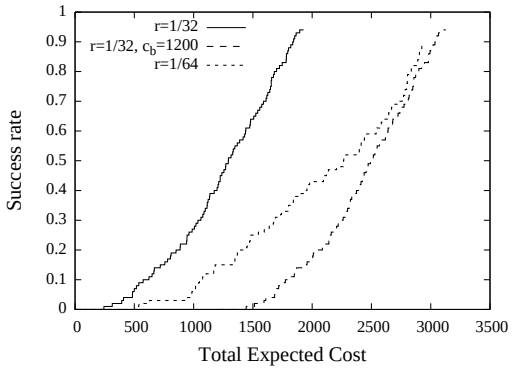
Suppose we are given a problem with an endgame of volume μ_0. Forward planning has a certain success rate profile $F(\mu_0, n)/C(\mu_0, n)$ as a function of the number of iterations. Suppose we are given the option to expand the endgame region to a larger volume μ_1. Doing so makes the success rate profile more favorable, but incurs cost c_e. Expansion is optimal if

$$\max_n \frac{F(\mu_1, n)}{C(\mu_1, n) + c_e} > \max_m \frac{F(\mu_0, m)}{C(\mu_0, m)}$$

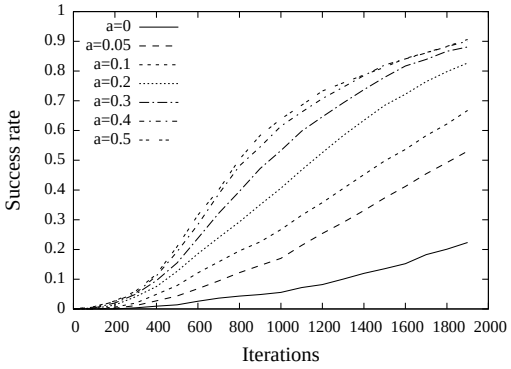
This can be interpreted visually by shifting the graph of $F(\mu_1, n)$ against $C(\mu_1, n)$ to the right by c_e units, and comparing the maximum slope of the two graphs. The



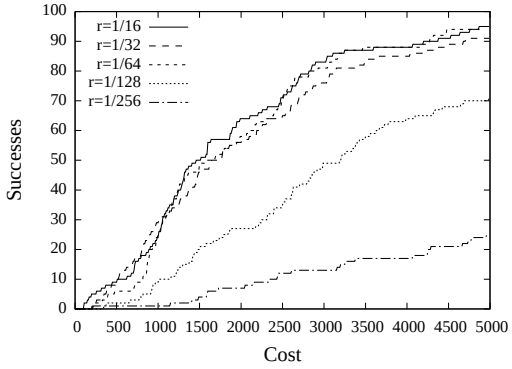
(a) Profiles of forward planning with endgames of different radii.



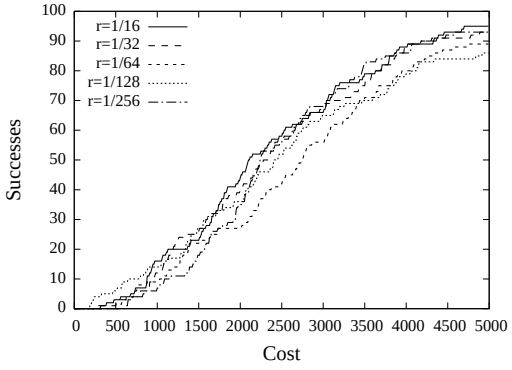
(b) An expansion cost of 1200 is acceptable to double the endgame of the radius.



(c) Profile $F(m, n)$ of bidirectional planning with a known goal configuration. The profile is projected along total iterations $k = n+m$, with slices along various values of $a = n/(m+n)$.



(d) Profiles of bidirectional planning with endgames of different radii.



(e) Profiles of bidirectional planning, with faster goal configuration sampling.

Figure 5.8:

optimal choice maximizes this slope. For example, consider the experiment of Figure 5.8a. If the planner is given the option to expand the endgame from 1/64 to 1/32 at cost c_e, then it should take this option if c_e is less than approximately 1200 (Figure 5.8b).

5.7.2 Bidirectional Planning Model

Bidirectional planning can be interpreted as forward planning with a growing endgame region. Denote the forward planning tree T_s. Each backward planning step extends a tree T_g rooted in $\mathcal{G}$. The planner can terminate when a node in T_s lies in $\mathcal{G} \cup \text{VIS}(T_g)$, where VIS denotes the visibility set of T_g. Here, $\mathcal{G} \cup \text{VIS}(T_g)$ acts as the expanded endgame region for forward planning. Backwards planning grows the endgame region.

Give the planner three choices: 1) sample a configuration, and if it lies in the endgame, use it to root a new tree in T_g; 2) expand an existing tree in T_s; and 3) expand an existing tree in T_g. This section rephrases the original question, "should a bidirectional planner ever expand the endgame?"

The cost of adding a tree c_t is geometrically distributed with parameter equal to the probability of sampling a feasible configuration. The parameter estimate might change as more samples are drawn.

Suppose we have a success rate profile for bidirectional planning, assuming we already have a rooted goal configuration. Index it by the number of start and goal iterations m and n. This gives a two dimensional profile $F(m, n)$ (see Figure 5.8c). Thus, given a history of expansions, we can pick the optimal tree to expand accordingly. (In the absence of other information, bidirectional planning is fastest when the start and goal trees have the same size.)

Let T_g^k be a tree in T_g. Let Z^k denote the event T_g^k lies in the same component of $\mathcal{F}$ as q_s. Let Z be the random variable denoting the fraction of $\mathcal{G}$ contained in the same component as q_s.

Assume $\mathcal{G}$ is small enough so that if multiple trees are rooted the same component as q_s, then expansion behaves almost identically to having a single tree per component. Suppose that T_s has m_0 nodes. Let n_0 denote the number of nodes of T_g in the same

component as q_s. Let p be the probability that backward expansion expands a tree k such that Z^k is true. n_0 can be estimated from Z_k and the sizes of each tree. p can be estimated from the Z^k, and takes different forms depending on the backward expansion mechanism.

The cost rate profile of expanding the start by m and the goal by n is

$$E\left[p\frac{F(m+m_0,n+n_0)-F(m_0,n_0)}{(1-F(m_0,n_0))(C(m+m_0,n+n_0)-C(m_0,n_0))}\right]$$

where the expectation is taken over all labels Z_k. The planner should compare this to the profile of sampling a new root in the endgame, then expanding the start by m and the goal by n:

$$E\left[p\frac{F(m+m_0,n+n_0)-F(m_0,n_0)}{(1-F(m_0,n_0))(c_t+C(m+m_0,n+n_0)-C(m_0,n_0))}\right]$$

where the expectation is taken over Z_k and c_t, and supposing that we have included a new root configuration in T_g.

Some special cases are as follows. If it is highly likely that the endgame region only contains one connected component, then it is highly unlikely for the optimal strategy to try to sample more than one endgame configuration. If the endgame volume is zero, but there is still a nonzero chance of sampling an endgame configuration, then the optimal strategy will sample an endgame configuration first.

5.7.3 Experiments

Figure 5.8c plots the success rate of experiments (again, on problem D of Figure 5.4a) using the small-endgame strategy outlined above. Here, potential endgame configurations were sampled from the entire space. Configuration feasibility tests were assigned cost 0.1, and path segment feasibility test had cost 1. Comparing the plots for the three smallest endgame radii with those of forward planning (Figure 5.8a), we see that, for a given cost, the bidirectional strategy is more likely to generate a feasible path.

The benefits are more dramatic when the planner can somehow localize the

endgame region. In the experiments of Figure 5.8d, potential endgame configurations were sampled from a region whose volume is 25 times that of the endgame's. This greatly reduces the cost of rooting a configuration in the endgame region, and makes performance nearly independent of endgame size.

5.8 Multi-Modal Planning with Bottlenecks

Consider RANDOM-MMP, the tree-growing multi-modal planning used in the manipulation planner of Section 4.2. A basic forward search implementation grows the tree by randomly sampling mode switches. But a faster backwards search described in Section 4.2 sampled configurations at "bottlenecks" – low-probability transitions several steps ahead – and then planned backwards. Backwards search has lower branching factor (there are in fact no branches), and the traditional intuition would cite the branching factor as the main reason why backwards search works better.

This section shows that this explanation is not entirely correct. Rather, backwards search works better because the feasibility of leaf transitions is highly *correlated* with the existence of a feasible path. Forward search is actually better if the correlations are weak or early mode switches are expensive (because the cost of early switches is amortized over later ones). We also describe how to pick the optimal branching factor for forward search.

5.8.1 Decision Theoretic Model

The planner of Section 4.2 used a three-level backwards planner, with an objective of finding high-quality, long-distance pushes. For a clearer exposition, this section analyzes a simpler 2-level tree problem (Figure 5.9a). The results should extend to three-levels and higher. Here each vertex represents a transition configuration, each edge represents a single mode plan to an adjacent mode, and the objective is to connect the root to a leaf (without a quality metric). We assume edges are tested exactly (by setting a time limit on single-mode PRM planning, after which an edge is marked invalid) with known expected cost and probability of success. We assume

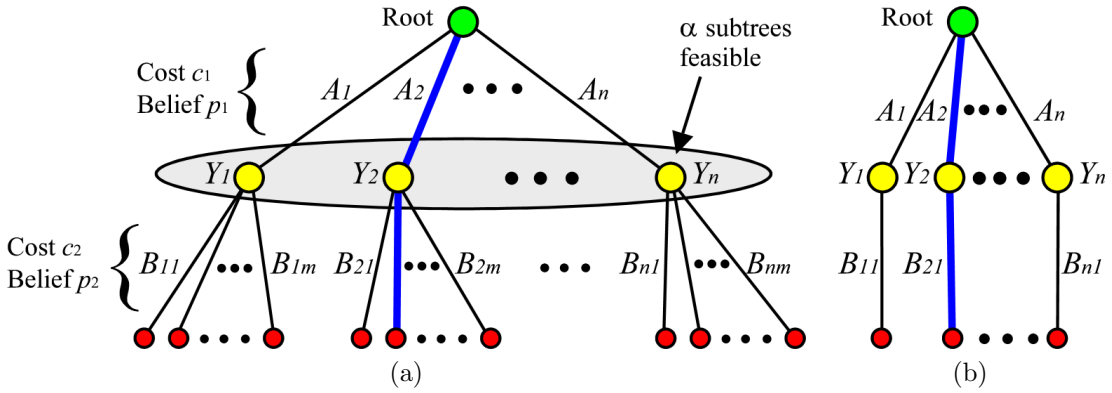


Figure 5.9: (a) A two-level tree built using forward search. (b) Backward search.

continuous mode sets, so that given a vertex, we may generate candidate children ad infinitum.

We compare two strategies:

1. *Forward search.* We generate level-1 edges from the root q_0 and test them for validity. Upon finding a valid single-mode path to q_1, we generate m level-2 edges from q_1 and test them for validity. Upon success, we return the path $q_0 \rightsquigarrow q_1 \rightsquigarrow q_2$. On failure, we continue generating level-1 edges.

2. *Non-branching backward search.* We generate a feasible final configuration q_2, then uniquely determine a q_1 at a mode adjacent to both q_0 and q_2. We then plan the level-2 path $q_1 \rightsquigarrow q_2$, and if it succeeds, we plan the level-1 path $q_0 \rightsquigarrow q_1$. If one of these two steps fails, we continue generating final configurations.

Figure 5.9a and Figure 5.9b illustrate the search trees built after n iterations of forward search and backward search, respectively. Let the level-1 edges be denoted A_i, for $i = 1, 2, \ldots, n$ and let the level-2 edges be denoted B_{ij} for $j = 1, 2, \ldots, m$. Suppose the level-1 and level-2 edges are tested with costs c_1 and c_2, respectively. Denote the probability of drawing a feasible random level-1 edge as $Pr(A_i) = p_1$, a random level-2 edge as $Pr(B_{ij}) = p_2$, and a random path as $Pr(A_i, B_{ij}) = p_3$. Also, define the random variable $Y_i = B_{i1} \vee B_{i2} \vee \cdots$ to represent the feasibility of a subtree.

Backward search is easy to analyze. Each iteration has probability p_3 of producing

a path to the goal, and has cost $c_2 + p_2 c_1$. As the number of iterations grow, the overall cost approaches $\frac{c_2+p_2c_1}{p_3}$.

The cost of forward search is affected not only by edge cost and success probability, but also by the distribution of feasible level-2 edges. We study variations in distribution using a two-parameter model. A parameter α defines the fraction of valid level-1 edges that yield feasible subtrees. A parameter β describes the ratio of p_3 to $p_1 p_2$. The fraction of feasible edges in feasible subtrees is given by $p_2\beta/\alpha$.

We have the following relations between variables:

- $Pr(A_i \wedge B_{ij}) = \beta p_1 p_2$
- $Pr(Y_i|A_i) = \alpha.\ \alpha \geq p_2\beta$
- $Pr(B_{ij}|A_i) = \beta Pr(B_{ij}).\ \beta p_1 \leq 1,\ \beta p_2 \leq 1.$

Roughly speaking, α describes how widely feasible subtrees are distributed, and β is a bias parameter that encodes correlation between level-1 and level-2 feasibility. An unbiased distribution sets $\beta = 1$.

5.8.2 Optimizing the Forward Search Strategy

In forward search, we must also select the number of level-2 edges m. Let $F(m)$ be the success rate profile of the level-2 sampling to test if Y_i is feasible (given that A_i is feasible), and $C(m)$ be the expected cost. Then the overall cost of forward search is $(c_1 + p_1 C(m))/(p_1 F(m))$. The profiles $F(m)$ and $C(m)$ are as follows:

$$F(m) = \alpha(1 - (1 - p_2\beta/\alpha)^m) \tag{5.6}$$

$$C(m) = c_2((1-\alpha) * m + \alpha^2/(p_2\beta) * (1 - (1 - p_2\beta/\alpha)^m)) \tag{5.7}$$

The optimal value of m can be found using discrete search, and rises sharply as α increases. If $\alpha = 1$, then the optimal m is infinite (the second-level edges are essentially coin flips), and forward search is always better than backwards search.

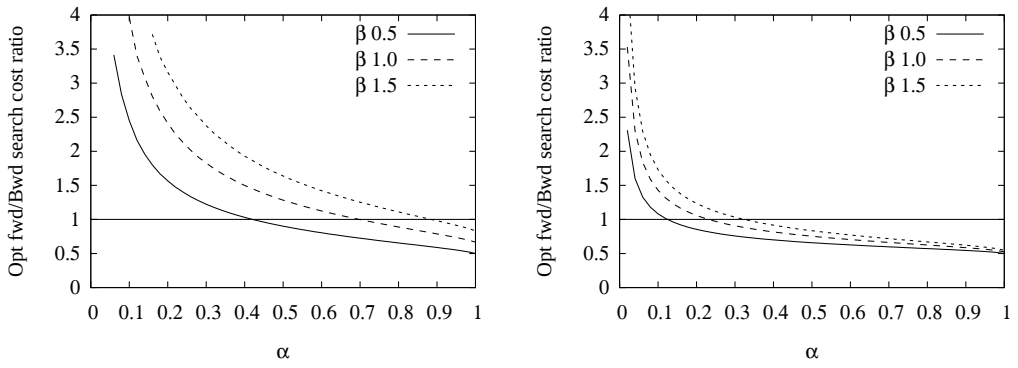


Figure 5.10: Comparing the cost ratio of forward to backward search for various α and β, for problems with (a) $p_2 = 0.1$ and (b) $p_2 = 0.01$.

5.8.3 Experimental Comparisons

Figure 5.10.a compares the ratio of the forward strategy cost to the backward strategy cost for a problem with $c_1 = 5$, $c_2 = 1$, $p_1 = 0.5$, and $p_2 = 0.1$. Forward search with $m = 1$ is over 3.5 times as expensive as backwards search. But when m is picked optimally, forward search is more expensive only when α is low – that is, feasible subtrees are hard to find. For distributions with high β (where level-2 and level-1 feasibility are highly correlated) backward search works better for a larger range of α.

These effects are even more pronounced when p_2 is decreased (Figure 5.10.a). For the backward strategy to be worthwhile, α must be lower – that is, feasible subtrees must be even harder to find.

These experiments show that the distribution of feasible subtrees (α and β) have large effects on the relative cost of forward search. These effects are greater than or comparable to the effects of changing other parameters, such as single-mode plan success rate and cost.

Chapter 6

Conclusion

This thesis addressed motion planning for legged robots navigating rugged outdoor terrain, and for object manipulation with a humanoid robot. Motion planning is necessary for robot autonomy, and is useful for human operator assistance and mechanism design in simulation. But its difficulty rises quickly as robot mechanisms become more complex, and their expected capabilities become more sophisticated. Legged locomotion and manipulation planning are difficult because these robots move in a high dimensional configuration space, which is decomposed into overlapping submanifolds of varying dimensionality, and each submanifold is additionally subject to complex geometric constraints. The work in this thesis addressed planner development in three general areas:

- Analyzing the multi-modal structure of these problems, and developing theoretically sound algorithms to solve them.
- Using domain knowledge in specially-engineered but widely-applicable motion strategies.
- Optimizing the planner's choice of subproblems and algorithm parameters using a decision theoretic approach.

This thesis has made contributions in each area.

6.1 Summary of Contributions

My contributions are as follows:

- Devised the problem-independent, roadmap-based multi-modal planning algorithm Multi-Modal-PRM (Section 2.3) and proved its theoretical completeness properties (Section 2.5).

- Devised the Incremental-MMPRM algorithm (Section 2.4) by combining Multi-Modal-PRM, a "lazy" search among feasible transitions (adopted from [21]), and a "fuzzy" search (adopted from [101]). This technique has the same completeness properties as Multi-Modal-PRM, but is faster by orders of magnitude.

- Implemented a legged locomotion planner that works on a wide range of robots on rough and steep terrain (Chapter 3). This was originally based on Tim Bretl's work on a 2D rock-climbing robot [21]. Extending this work to 3D robots in rough terrain required required the following contributions:

  - System modeling, such as contact modeling and torque limit checking.
  - Better planning subroutines, such as a numerical method to reduce the rejection rate of configuration samplers (Section 3.3.3), geometric methods to prune infeasible transitions (Section 3.3.5), and new search heuristics (Section 3.3.6).
  - Improved reliability by integrating the probabilistically complete Incremental-MMPRM algorithm.

- Developed the technique that biases a legged locomotion planner with motion primitives to make motion more natural (Section 4.1). This was joint work with Tim Bretl, Kensuke Harada, and Jean-Claude Latombe.

- Developed a manipulation planner for a humanoid robot, with domain knowledge consisting of a utility precomputation (Section 4.2). This was joint work with Victor Ng-Thow-Hing and Hector Gonzalez-Baños.

- Applied a principled decision-theoretic approach to optimize several planning subroutines, including constraint testing, path shortening, perturbation-based configuration sampling, and fuzzy search ordering (Chapter 5).

6.2 Future Research

Multi-modal planning analysis This thesis proved a completeness result for Multi-Modal-PRM assuming that the system's set of modes was already discretized. If the planner fails on a given discretization, the discretization may have been poor. Future work should address developing an algorithm for continuous-mode systems which has theoretical completeness guarantees.

Treating uncertainty during planning The planners in this thesis assume a perfect model of the environment and robot. This assumption does not hold under limited information (such as sensor occlusions or unknown friction coefficients), sensing uncertainty (such sensor noise), or motion uncertainty (such as deforming terrain or external disturbances). Periodic replanning can use new sensor information as it becomes available and correct for some amount of drift (as in [67]). Using a reactive controller in conjunction with the planner could help the robot recover from moderate disturbances (as in [45]). Here, a planner can be used as part of a larger system that integrates sensing, long-term multi-modal planning, and short-term reactive hybrid control. However, other types of uncertainty may result in unrecoverable or catastrophic failure, and cannot be corrected for reactively. Given the complexity of legged robot systems, a thorough treatment of uncertainty seems enormously intractable (at least with current tools, such as Markov decision processes). But it may be possible to anticipate a few crucial situations that merit contingency planning, which may be a promising area for improving a legged robot's reliability in precarious situations.

Planning with Dynamics The planners in this thesis assume quasistatic dynamics. As remarked in Chapter 3, a postprocessing step can convert the output of the planner into a dynamically feasible path. But postprocessing cannot produce

highly dynamic motions like running or jumping. It may also have difficulty producing highly energy efficient motions that take advantage of passive dynamics. Dynamic constraints might be addressed in a multi-modal planner simply by using single-mode planners that handle nonholonomic constraints. Some sample-based kinodynamic planners have been proven to converge [67], but very high dimensional configuration spaces (say, above 10-D) are still out of the realm of practicality. Motion primitives (Chapter 4) might be able provide heuristics to make these planners practical for legged locomotion planning. Another promising alternative might use well-designed multi-contact control strategies [106] for single-step planning. This technique would also help in the treatment of uncertainty.

Making planning faster The planners in this work, take tens of minutes on a PC, which is still too slow for some purposes. Interactive rates could be achieved by powerful supercomputers (PRM planners and search algorithms are easily parallelized [1, 42]), which could work on a remotely operated robot (like ATHLETE). However, a supercomputer would be too bulky, heavy, and power-hungry for a battery-powered autonomous robot (like HRP-2). Future work could improve the planner's speed even further by refining the methods presented here. Improving planning speed is especially necessary to incorporate dynamics into planning.

Treating planning as a limited resource In complex systems like legged locomotion and manipulation, planning will not likely be simultaneously reliable, cheap, and high quality. In an integrated system that interleaves planning and execution, should the robot spend time making a detailed plan, or does a coarse one suffice? Should it spend time improving motion quality through optimization, or just execute an unnatural motion? In part, these questions involve the psychology of human-robot interaction, and could lead to interesting usability studies.

Automating decision-theoretic optimization The optimal subproblem execution strategies in Chapter 5 are currently derived by hand, but could be automated. Though solving POMDPs is exceptionally hard, approximate solvers (e.g, [73]) may

be able to generate reasonably good strategies. Taking this idea one step further, we could automate the process of choosing and learning models of background knowledge. This would significantly reduce the effort needed to optimize motion planners.

Appendix A

Probabilistic Roadmaps and Expansiveness

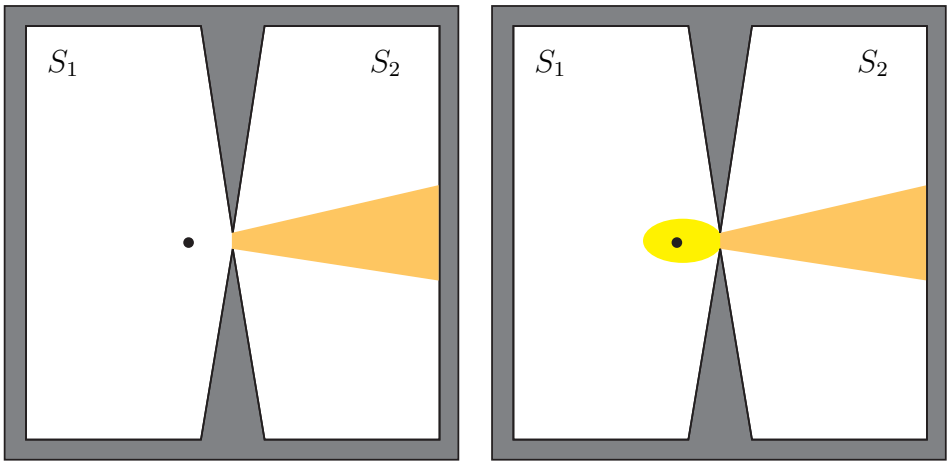
This appendix reviews PRM planners and the theoretical completeness properties of a basic PRM planner in expansive spaces.

A.1 Basic Algorithm

PRM planners address the problem of connecting two configurations q_s and q_g in the feasible space $\mathcal{F}$, a subset of configuration space $\mathcal{Q}$. Though it is prohibitively expensive to compute an exact representation of $\mathcal{F}$, feasibility tests are usually cheap. So to approximate the connectivity of $\mathcal{F}$, PRM planners build a *roadmap* $\mathcal{R}$, a network of feasible configurations (called *milestones*) connected with straight-line segments. A basic PRM planner operates as follows:

BASIC-PRM(q_{start}, q_{goal}, n)
Add q_{start} and q_{goal} to $\mathcal{R}$ as milestones.
Repeat n times:

1. Sample a configuration q uniformly from $\mathcal{Q}$, and test its feasibility. Repeat until a feasible sample is found.



(a) (b)

Figure A.1: A poorly expansive space. (a) The visibility set $\mathcal{V}(q)$ in region S_2, of a point q in S_1. (b) $\textsc{Lookout}_\beta(S_1)$ is the set of points that see at least a β fraction of S_2.

2. Add q to $\mathcal{R}$ as a new milestone. Connect it to nearby milestones q' in $\mathcal{R}$ if the line segment between q and q' lies in $\mathcal{F}$.

If $\mathcal{R}$ contains a path between q_{start} and q_{goal}, return the path.
Otherwise, return 'failure'.

If a PRM planner produces a path successfully, the path is guaranteed to be feasible, but if it fails, then we cannot tell whether no path exists or the cutoff n was set too low.

A.2 Performance in Expansive Spaces

Basic-PRM and several variants have been shown to be probabilistically complete, that is, the probability of incorrectly returning failure approaches 0 as n increases. One particularly strong completeness theorem proves that PRMs converge exponentially, given that $\mathcal{F}$ is *expansive* [64].

The notion of expansiveness expresses the difficulty of constructing a roadmap

that captures the connectivity of $\mathcal{F}$. The success of PRMs in high dimensional spaces is partially explained by the fact that expansiveness is not explicitly dependent on the dimensionality of $\mathcal{F}$. Let $\mu(S)$ measure the volume of any subset $S \subseteq \mathcal{F}$ (with $\mu(\mathcal{F})$ finite), and let $\mathcal{V}(q)$ be the set of all points that can be connected to q with a straight line in $\mathcal{F}$. The *lookout* set of a subset S of $\mathcal{F}$ is defined as the subset of S that can "see" a subtantial portion of the complement of S. (see Figure A.1). Formally, given a constant $\beta \in (0, 1]$ and a subset S of a connected component $\mathcal{F}'$ in $\mathcal{F}$, define

$$\textsc{Lookout}_\beta(S) = \{q \in S \quad | \quad \mu(\mathcal{V}(q) \setminus S) \geq \beta\mu(\mathcal{F}' \setminus S)\}$$

For constants ϵ, α, $\beta \in (0, 1]$, $\mathcal{F}$ is said to be $(\epsilon, \alpha, \beta)$-*expansive* if:

1. For all $q \in \mathcal{F}$, $\mu(\mathcal{V}(q)) \geq \epsilon\mu(\mathcal{F})$.

2. For any connected subset S, $\mu(\textsc{Lookout}_\beta(S)) \geq \alpha\mu(S)$.

The first property is known as ϵ-goodness, and states that each configuration "sees" a significant fraction of $\mathcal{F}$. The second property can be interpreted as follows. View S as the visibility set of a single roadmap component $\mathcal{R}'$. Let $\mathcal{F}'$ be the component of feasible space in which S lies. Then, with significant probability (at least $\alpha\mu(S)$), a random configuration will simultaneously connect to $\mathcal{R}'$ and significantly reduce the fraction of $\mathcal{F}'$ not visible to $\mathcal{R}'$ (by at least β).

The primary convergence result of [64] can be restated as follows:

Theorem A.2.1. *If $\mathcal{F}$ is* $(\epsilon, \alpha, \beta)$-expansive*, then the probability that a roadmap of n uniformly, independently sampled milestones fails to connect q_{start} and q_{goal} is no more than ce^{-dn} for some positive constants c and d.*

The constants c and d are simple functions of ϵ, α, and β. If $\mathcal{F}$ is favorably expansive (ϵ, α, and β are high), the bound is close to zero, and Basic-PRM will find a path q_{start} and q_{goal} relatively quickly. If, on the other hand, $\mathcal{F}$ is poorly expansive (ϵ, α, and β are low), then PRM performance might be poor for certain query configurations q_{start} and q_{goal}. A complementary theorem proven in [63] states that a PRM planner will succeed with arbitrarily low probability for any fixed n in spaces with small α and β.

Bibliography

[1] Mert Akinc, Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, Erion Plaku, and Lydia E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *Int. Symp. Rob. Res.*, Siena, Italy, 2003.

[2] R. Alami, J.-P. Laumond, and T. Siméon. Two manipulation planning algorithms. In K. Goldberg, D. Halperin, Jean-Claude Latombe, and R. Wilson, editors, *Alg. Found. Rob.*, pages 109–125. A K Peters, Wellesley, MA, 1995.

[3] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE Trans. Robot. and Autom.*, volume 16, pages 442–447, 2000.

[4] Manuel Armada, Pablo Gonzalez de Santos, María A. Jiménez, and Manuel Prieto. Application of CLAWAR machines. *Int. J. Rob. Res.*, 22(3-4):251–264, 2003.

[5] K.S. Arun, T. Huang, and S.D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Machine Intell.*, 9(5):698–700, 1987.

[6] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.

[7] John E. Bares and David S. Wettergreen. Dante II: Technical description, results and lessons learned. *Int. J. Rob. Res.*, 18(7):621–649, 1999.

[8] J. Barraquand and J.C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-3-4):121–155, 1993.

[9] D.A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, 1985.

[10] D. Bevly, S. Farritor, and S. Dubowsky. Action module planning and its application to an experimental climbing robot. In *IEEE Int. Conf. Rob. Aut.*, pages 4009–4014, 2000.

[11] J.E. Bobrow, S . Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *Int. J. Rob. Res.*, 4(3), 1985.

[12] J.E. Bobrow, B. Martin, G. Sohl, E.C. Wang, F.C. Park, and Junggon Kim. Optimal robot motions for physical criteria. *J. of Robotic Systems*, 18(12):785–795, 2001.

[13] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy prm. In *IEEE Int. Conf. Rob. Aut.*, pages 521–528, San Francisco, CA, 2000.

[14] J.-D. Boissonnat, O. Devillers, L. Donati, and F. Preparata. Motion planning of legged robots: The spider robot problem. *Int. J. of Computational Geometry and Applications*, 5(1-2):3–20, 1995.

[15] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Int. Conf. Rob. Aut.*, pages 1018–1023, 1999.

[16] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[17] Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[18] Timothy Bretl. *Multi-Step Motion Planning: Application to Free-Climbing Robots*. PhD thesis, Stanford University, 2005.

[19] Timothy Bretl. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *Int. J. Rob. Res.*, 25(4):317–342, 2006.

[20] Timothy Bretl and Sanjay Lall. A fast and adaptive test of static equilibrium for legged robots. In *IEEE Int. Conf. Rob. Aut.*, Orlando, FL, 2006.

[21] Timothy Bretl, Sanjay Lall, Jean-Claude Latombe, and Stephen Rock. Multi-step motion planning for free-climbing robots. In *WAFR*, Zeist, Netherlands, 2004.

[22] Timothy Bretl, Jean-Claude Latombe, and Stephen Rock. Toward autonomous free-climbing robots. In *Int. Symp. Rob. Res.*, Siena, Italy, 2003.

[23] Timothy Bretl, Stephen Rock, Jean-Claude Latombe, Brett Kennedy, and Hrand Aghazarian. Free-climbing with a multi-use robot. In *Int. Symp. Exp. Rob.*, Singapore, 2004.

[24] Leoncio Briones, Paul Bustamante, and Miguel A. Serna. Wall-climbing robot for inspection in nuclear power plants. In *IEEE Int. Conf. Rob. Aut.*, pages 1409–1414, San Diego, CA, 1994.

[25] M. Buehler, U. Saranli, D. Papadopoulos, and D. Koditschek. Dynamic locomotion with four and six-legged robots. In *Int. Symp. on Adaptive Motion of Animals and Machines*, Montreal, Canada, 2000.

[26] Francesco Bullo and Miloš Žefran. On modeling and locomotion of hybrid mechanical systems with impacts. In *IEEE Conf. on Decision and Control*, pages 2633–2638, Tampa, FL, 1998.

[27] Francesco Bullo and Miloš Žefran. Modeling and controllability for a class of hybrid mechanical systems. *IEEE Trans. Robot. and Autom.*, 18(4):563–573, 2002.

[28] Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *IEEE Int. Conf. Rob. Aut.*, pages 3120–3125, Barcelona, Spain, 2005.

[29] Brendan Burns and Oliver Brock. Single-query entropy-guided path planning. In *IEEE Int. Conf. Rob. Aut.*, pages 2124–2129, Barcelona, Spain, 2005.

[30] Brendan Burns and Oliver Brock. Single-query motion planning with utility-guided random trees. In *IEEE Int. Conf. Rob. Aut.*, 2007.

[31] R.R. Burridge, A.A. Rizzi, and D.E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *Int. J. Rob. Res.*, 18(6):534–555, 1999.

[32] A. Casal. *Reconfiguration Planning for Modular Self-Reconfigurable Robots*. PhD thesis, Stanford University, Stanford, CA, 2001.

[33] A. Casal and M. Yim. Self-reconfiguration planning for a class of modular robots. In *SPIE II*, pages 246–257, 1999.

[34] S. Chaudhuri and V. Koltun. Smoothed analysis of probabilistic roadmaps. In *Fourth SIAM Conference of Analytic Algorithms and Computational Geometry*, 2007.

[35] M. Cherif and K. Gupta. Planning for in-hand dextrous manipulation. In P.K. Agarwal, L.E. Kavraki, and M.T. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 103–117. AK Peters, Ltd., 1998.

[36] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. In *IEEE Int. Conf. Hum. Rob.*, Munich, Germany, 2003.

[37] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[38] V. A. Cicirello and S. F. Smith. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *AAAI*, 2005.

[39] J. Cortés and T Siméon. Sampling-based motion planning under kinematic loop-closure constraints. In *WAFR*, Zeist, Netherlands, 2004.

[40] J. Cortés, T. Siméon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains using prm methods. In *IEEE Int. Conf. Rob. Aut.*, Washington, D.C., 2002.

[41] C. Eldershaw and M. Yim. Motion planning of legged vehicles in an unstructured environment. In *IEEE Int. Conf. Rob. Aut.*, Seoul, South Korea, 2001.

[42] Matthew Evett, James Hendler, and Dana Nau. Pra*: massively parallel heuristic search. *Journal of Parallel and Distributed Computing*, 25:133–143, 1995.

[43] Pierre Ferbach and Jérôme Barraquand. A method of progressive constraints for manipulation planning. *IEEE Trans. Robot. and Autom.*, 13(4):473–485, 1997.

[44] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. Robot.*, 25(1):116–129, 2002.

[45] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *AIAA J. of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[46] V. Gavrilets, E. Frazzoli, B. Mettler, M. Peidmonte, and E. Feron. Aggressive maneuvering of small autonomous helicopters: A human-centered approach. *Int. J. Rob. Res.*, 20(10):795–807, 2001.

[47] Roland Geraerts and Mark Overmars. Clearance based path optimization for motion planning. In *IEEE Int. Conf. Rob. Aut.*, New Orleans, LA, 2004.

[48] Roland Geraerts and Mark H. Overmars. Creating high-quality paths for motion planning. *Intl. J. of Rob. Res.*, 26(8):845–863, 2007.

[49] P.E. Gill, W. Murray, and M.H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Addison Wesley, 1991.

[50] Michael Gleicher. Retargetting motion to new characters. In *SIGGRAPH*, pages 33–42, 1998.

[51] Bill Goodwine and Joel Burdick. Motion planning for kinematic stratified systems with application to quasi-static legged locomotion and finger gaiting. In *WAFR*, Hanover, NH, 2000.

[52] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. In *ACM SIGGRAPH*, pages 171–180, 1996.

[53] Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, 170:19–58, 2006.

[54] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531, 2004.

[55] Kris Hauser, Timothy Bretl, Kensuke Harada, and Jean-Claude Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *WAFR*, New York, NY, 2006.

[56] Kris Hauser, Timothy Bretl, and Jean-Claude Latombe. Learning-assisted multi-step planning. In *IEEE Int. Conf. Rob. Aut.*, Barcelona, Spain, 2005.

[57] Kris Hauser, Timothy Bretl, and Jean-Claude Latombe. Non-gaited humanoid locomotion planning. In *IEEE Humanoids*, Tsukuba, Japan, 2005.

[58] Kris Hauser, Timothy Bretl, Jean-Claude Latombe, and Brian Wilcox. Motion planning for a six-legged lunar robot. In *WAFR*, New York, NY, 2006.

[59] Kris Hauser, Victor Ng-Thow-Hing, and Hector Gonzales-Ba nos. Multi-modal planning for a humanoid manipulation task. In *Intl. Symposium on Robotics Research*, Hiroshima, Japan, 2007.

[60] Shigeo Hirose, Kan Yoneda, and Hideyuki Tsukagoshi. Titan VII: Quadruped walking and manipulating robot on a steep slope. In *IEEE Int. Conf. Rob. Aut.*, pages 494–500, Albuquerque, NM, 1997.

[61] C. Holleman and L. Kavraki. A framework for using the workspace medial axis in prm planners. In *IEEE Int. Conf. Rob. Aut.*, volume 2, pages 1408–1413, 2000.

[62] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE Int. Conf. Rob. Aut.*, Taipei, Taiwan, 2003.

[63] D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Rob. Res.*, 25(7):627–643, 2006.

[64] D. Hsu, Jean-Claude Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE Int. Conf. Rob. Aut.*, pages 2219–2226, 1997.

[65] D. Hsu, G. Snchez-Ante, and Z. Sun. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *IEEE Int. Conf. Rob. Aut.*, pages 3885–3891, Barcelona, Spain, 2005.

[66] David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin. On finding narrow passages with probabilistic roadmap planners. In *WAFR*, pages 141–153, Natick, MA, 1998.

[67] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Kinodynamic motion planning amidst moving obstacles. *Int. J. Rob. Res.*, 21(3):233–255, Mar 2002.

[68] Y. K. Hwang, S. C. Kang, S. Lee, S. M. Park, K. R. Cho, H. S. Kim, and C. W. Lee. Human interface, automatic planning, and control of a humanoid robot. *International Journal of Robotics Research*, 17(11):1131–1149, 1998.

[69] S. Kagami, F. Kanehiro, Y. Tmiya, M. Inaba, and H. Inoue. Autobalancer: An online dynamic balance compensation scheme for humanoid robots. In *WAFR*, pages 329–340, 2000.

[70] Kenji Kaneko, Fumio Kanehiro, Shuuki Kajita, Hirohisa Hirukawa, Toshikazu Kawasaki, Masaru Hirata, Kazuhiko Akachi, and Takakatsu Isozumi. Humanoid robot HRP-2. In *IEEE Int. Conf. Rob. Aut.*, pages 1083–1090, New Orleans, LA, 2004.

[71] L. E. Kavraki, P. Svetska, Jean-Claude Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. and Autom.*, 12(4):566–580, 1996.

[72] R.B. Kearfott. Interval analysis: Verifying feasibility. *Encyclopedia of Optimization*, 3:43–45, 2001.

[73] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[74] A. Kheddar, D. Atlani, A. Iles, and P. Blazevic. New trends in legged robots teleoperation. In *IEEE International Workshop on Robot and Human Communication*, pages 232–237, 1996.

[75] D.E. Koditschek. Robot assembly: another source of nonholonomic control problems. In *American Controls Conference*, pages 1627–1632, Boston, MA, 1991.

[76] Yoshihito Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *IEEE Int. Conf. Rob. Aut.*, pages 945–952, San Diego, CA, 1994.

[77] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH*, pages 473–482, San Antonio, Texas, 2002.

[78] Taesoo Kron and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 29–38, Los Angeles, CA, 2005.

[79] E. Krotkov and R. Simmons. Perception, planning, and control for autonomous walking with the ambler planetary rover. *Int. J. Rob. Res.*, 15:155–180, 1996.

[80] James J. Kuffner, Jr. *Autonomous Agents for Real-Time Animation.* PhD thesis, Stanford University, 1999.

[81] James J. Kuffner, Jr., S. Kagami, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. In *First Int. Conf. on Humanoid Robotics*, Boston, MA, 2000.

[82] James J. Kuffner, Jr., Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Footstep planning among obstacles for biped robots. In *IEEE/RSJ Int. Conf. Int. Rob. Sys.*, Maui, HI, 2001.

[83] James J. Kuffner, Jr., Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Motion planning for humanoid robots. In *Int. Symp. Rob. Res.*, Siena, Italy, 2003.

[84] Jean-Paul Laumond. Finding collision-free smooth trajectories for a non-holonomic mobile robot. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1120–1123, 1987.

[85] J.P. Laumond, P. Jacobs, M. Taix, and R. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. and Autom.*, 10(5):577–593, 1994.

[86] S. M. LaValle and James J. Kuffner, Jr. Rapidly-exploring random trees: progress and prospects. In *WAFR*, 2000.

[87] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[88] Steven M. LaValle, Jeffery H. Yakey, and Lydia E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *IEEE Int. Conf. Rob. Aut.*, Detroit, MI, 1999.

[89] C.T. Lawrence, J.L. Zhou, and A.L. Tits. User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical Report TR-94-16r1, 20742, Institute for Systems Research, University of Maryland, College Park, MD, 1997.

[90] Tsai-Yen Li, Pei-Feng Chen, and Pei-Zhi Huang. Motion planning for humanoid walking in a layered environment. In *IEEE International Conference on Robotics and Automation*, pages 3421–3427, 2003.

[91] Lin Liao, Dieter Fox, and Henry Kautz. Location-based activity recognition. In *Advances in Neural Information Processing Systems*, 2005.

[92] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, 2005.

[93] K. Lynch and M. Mason. Stable pushing: Mechanics, controllability, and planning. *The Int. J. Rob. Res.*, 15(6):533–556, Dec 1996.

[94] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI/IAAI*, pages 541–548, 1999.

[95] M. Mason, E. Rimon, and J. Burdick. The stability of heavy objects with multiple contacts. In *IEEE Int. Conf. Rob. Aut.*, 1995.

[96] Michael Meredith and Steve Maddock. Adapting motion capture data using weighted real-time inverse kinematics. *Comput. Entertain.*, 3(1), 2005.

[97] T.G. Miller, T. Bretl, and S. Rock. Control of a climbing robot using real-time convex optimization. In *Proc. IFAC Symposium on Mechatronic Systems*, 2006.

[98] Patrycja E. Missiuro and Nicholas Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *IEEE Int. Conf. Rob. Aut.*, Orlando, FL, 2006.

[99] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Hybrid Systems: Computation and Control, LNCS 1790*, Mar 2000.

[100] R.S. Mosher and R.A. Liston. A versatile walking truck. In *Transp. Eng. Conf.*, pages 255–268, London, 1968.

[101] Christian L. Nielsen and Lydia E. Kavraki. A two level fuzzy prm for manipulation planning. In *IEEE/RSJ Int. Conf. Int. Rob. Sys.*, pages 1716–1721, Takamatsu, Japan, 2000.

[102] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars. An effective framework for path planning amidst movable obstacles. In *WAFR*, New York, NY, 2006.

[103] Yizhar Or and Elon Rimon. Computing 3-legged equilibrium stances in three-dimensional gravitational environments. In *IEEE Int. Conf. Rob. Aut.*, Orlando, FL, 2006.

[104] J.-S. Pang and J. Trinkle. Stability characterizations of rigid body contact problems with coulomb friction. *Z. Angew. Math. Mech.*, 80(10):643–663, 2000.

[105] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov chain decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[106] Jaheung Park and Oussama Khatib. Robot multiple contact control. In *Robotica*, 2008.

[107] Julien Pettré, Jean-Paul Laumond, and Thierry Siméon. A 2-stages locomotion planner for digital actors. In *Eurographics/SIGGRAPH Symp. Comp. Anim.*, 2003.

[108] Marko B. Popovic, Ambarish Goswami, and Hugh Herr. Ground reference points in legged locomotion: Definitions, biological trajectories and control implications. *Int. J. Rob. Res.*, 24(12):1013–1032, 2005.

[109] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *SIGGRAPH*, pages 11–20, 1999.

[110] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico, 1979.

[111] Liu Ren, Alton Patrick, Alexei A. Efros, Jessica K. Hodgins, and James M. Rehg. A data-driven approach to quantifying natural human motion. *ACM Trans. Graph.*, 24(3):1090–1097, 2005.

[112] A. Sahbani, J. Cortés, and T. Siméon. A probabilistic algorithm for manipulation planning under continuous grasps and placements. In *IEEE/RSJ Int. Conf. Int. Rob. Sys.*, pages 1560–1565, Lausanne, Switzerland, 2002.

[113] Gildardo Sánchez and Jean-Claude Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *Int. J. of Rob. Res.*, 21(1):5–26, 2002.

[114] F. Schwarzer, M. Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. In *WAFR*, Nice, France, Dec 2002.

[115] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *Int. J. Humanoid Robotics*, 2(4):505–518, 2005.

[116] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Trans. Graph.*, 20(2):67–94, 2001.

[117] T. Siméon, J. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. In *Int. Conf. on Intel. Robots and Systems*, 1999.

[118] M. Stilman. *Navigation Among Movable Obstacles*. PhD thesis, Carnegie Mellon University, 2007.

[119] R. Volpe, T. Estlin, S. Laubach, C. Olson, and J. Balaram. Enhanced mars rover navigation techniques. In *IEEE International Conference on Robotics and Automation*, volume 24, pages 926–931, San Francisco (USA), 2000.

[120] David Wettergreen. *Robotic walking in natural terrain*. PhD thesis, Carnegie Mellon University, 1995.

[121] David Wettergreen, Hans Thomas, and Chuck Thorpe. Planning strategies for the ambler walking robot. In *IEEE Int. Conf. on Systems Engineering*, pages 198–203, Pittsburgh, PA, 1990.

[122] G. Wilfong. Motion planning in the presence of movable obstacles. In *Fourth Annual Symposium on Computational Geometry*, pages 279 – 288, Urbana-Champaign, IL, 1988.

[123] R.H. Wilson and J.C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1995.

[124] Andrew Witkin and Zoran Popović. Motion warping. In *SIGGRAPH*, pages 105–108, Los Angeles, CA, 1995.

[125] J. Xu, T. J. Koo, and Z. Li. Finger gaits planning for multifingered manipulation. In *IEEE Conf. on Intel. Rob. and Sys.*, San Diego, CA, 2005.

[126] Jeffery H. Yakey, Steven M. LaValle, and Lydia E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Trans. Robot. and Autom.*, 17(6):951–958, 2001.

[127] Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph.*, 23(3):532–539, 2004.

[128] Masahito Yashima and Hideya Yamaguchi. Dynamic motion planning whole arm grasp systems based on switching contact modes. In *IEEE Int. Conf. Rob. Aut.*, Washington, D.C., 2002.

[129] R. Zhang. Design of a climbing robot: Capuchin. In *Proc. 5th Intl. Conf. on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS)*, Linz, Austria, June 2008.