Exploiting Structure: A Guided Approach to Sampling-Based Robot Motion Planning

A Dissertation Presented

by

Brendan Burns

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2007

Computer Science

EXPLOITING STRUCTURE: A GUIDED APPROACH TO SAMPLING-BASED ROBOT MOTION PLANNING

A Dissertation Presented

by

BRENDAN BURNS

Approved as to style and content by:

Oliver Brock, Chair

Andrew McCallum, Member

Andrew Barto, Member

Richard E.A. van Emmerik, Member

Bruce Croft, Department Chair
Computer Science

Acknowledgments

Graduate school is a long, convoluted and challenging process. I could never have done it alone. More people than I could possibly name have helped me along the way. But there are a few who deserve special recognition for their impact on my life.

First and foremost, my wonderful, beautiful and patient wife, Robin Sanders. Without your support, none of this would have been possible. Without your love, none of this would have been worthwhile.

My mother, Jody Burns, for her love, support and interest in my work. You laid the foundation for everything that I have accomplished in my life.

Megan and Andrew Ulland and the remainder of my family for their support and encouragement.

My adviser Oliver Brock, who has mentored me through this tangled process and taught me a great deal about many things. I'm also thankful to my former adviser, Paul Cohen, who started me down this path.

Many, many, others have also contributed to my academic development, including Brian Levine, Emery Berger, Mark Corner, David Westbrook, Andrea Danyluk and all of the professors who taught my classes. Everything I do in the future will reflect the things I've learned from you.

I would also like to thank my fellow graduate students, TJ Brunette, Yuandong Yang, Steve Hart, Dubi Katz and the rest of the laboratory for perceptual robotics as well as Brent Heeringa, Charles Sutton, Mike O'Neill, Matt Schmill and the other EKSL survivors.

Finally, I'd like to thank Rao's, Amherst Coffee, the Jones Library, the Frost Library and the DuBois Library for providing me space where I could get my work done.

Abstract

Exploiting Structure: A Guided Approach to Sampling-Based Robot Motion Planning

February 2007

Brendan Burns

B.A., Williams College

M.Sc., University of Massachusetts Amherst

Ph.D., University of Massachusetts Amherst

Directed by: Professor Oliver Brock

Robots already impact the way we understand our world and live our lives. However, their impact and use is limited by the skills they possess. Currently deployed autonomous robots lack the manipulation skills possessed by humans. To achieve general autonomy and applicability in the real world, robots must possess such skills. Autonomous manipulation requires algorithms that rapidly and reliably compute collision-free motion for robotic limbs with many degrees of freedom. Unfortunately, adequate algorithms for this task do not currently exist. Though there are many dimensions of the real-world planning task that require further research. A central problem of reliable real-world planning is that planners must rely on incomplete and inaccurate information about the world in which they are planning. The motion planning problem has exponential complexity in the robot's degrees of freedom. Consequently, the most successful planning algorithms use incomplete information obtained via sampling a subset of all possible movements. Additionally, real-world robots generally obtain information about the state of their environment through lasers, cameras and other sensors. The information obtained from these sensors contains noise and error. Thus the planner's incomplete information about the world is possibly inaccurate as well. Despite such limited information, a planner must be capable of quickly generating collision free motions to facilitate general purpose autonomous robots.

This thesis proposes a new *utility-guided* framework for motion planning that can reliably compute collision-free motions with the efficiency required for real-world planning. The utility-guided approach begins with the observation there is regularity in space of possible motions available to a robot. Further, certain motions are more crucial than others for computing collision free paths. Together these observations form structure in the robot's space of possible movements. This structure provides a guide for the planner's exploration of possible motions. Because a complete understanding of this structure is computationally intractable, the utility-guided framework incrementally develops an approximate model discovered by past exploration. This

model of the structure is used to select explorations that maximally benefit the planner. Information provided by each exploration improves the planner's approximation. The process of incremental improvement and further guided exploration iterates until an adequate model of configuration space is constructed. Discovering and exploiting structure in a robot's configuration space enables a utility-guided planner to achieve the performance and reliability required by real-world motion planning.

This thesis describes applications of the utility-guided motion-planning framework to multi-query sampling-based roadmap and and random-tree motion planning. Additionally, the utility-guided framework is extended to develop a planner that can successfully plan despite inaccuracies in its perception of the environment and to guide further sensing to reduce uncertainty and maximally improve the utility of the path.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

Robots increasingly impact our daily lives. Aibo [27], a robotic pet, lives in thousands of households. Jason II's [41] cameras provide dramatic footage of the ocean floor. Likewise, NASA's Spirit and Opportunity [77] rovers allow humans to see and explore the surface of Mars. More than ten autonomous vehicles drove more than 130 miles to complete the DARPA "Grand Challenge". Unmanned semi-autonomous vehicle, either in the air or underwater, are expanding our ability to monitor and understand the changing earth. Robots like Packbot [42] are replacing humans in dangerous environments like war zones and disaster zones. Robots have also replaced humans on the factory floor, assembling everything from automobiles to iPods, increasing productivity and decreasing costs. Robots are replacing humans in mundane tasks as well. The Roomba [43] vacuums floors and the several automobiles [28] can parallel park themselves. Figure 1.1 shows several of these robots.

But this impact, significant as it is, represents only the smallest subset of the real world tasks an autonomous robot might accomplish. This limited impact is due to the limitations of the robots themselves. Robots actually deployed today in the real world are purposefully engineered and built with significant limitations. This purposeful limitation of skills enables the robots to successfully achieve the task for which they were designed. Some robots, like the Roomba and Prius, are successful by limiting their focus to a single simple activity (vacuuming or parallel parking). In contrast, factory robots are successful in a sophisticated assembly task by operating in a tightly constrained environment. Robots which are capable of multiple sophisticated

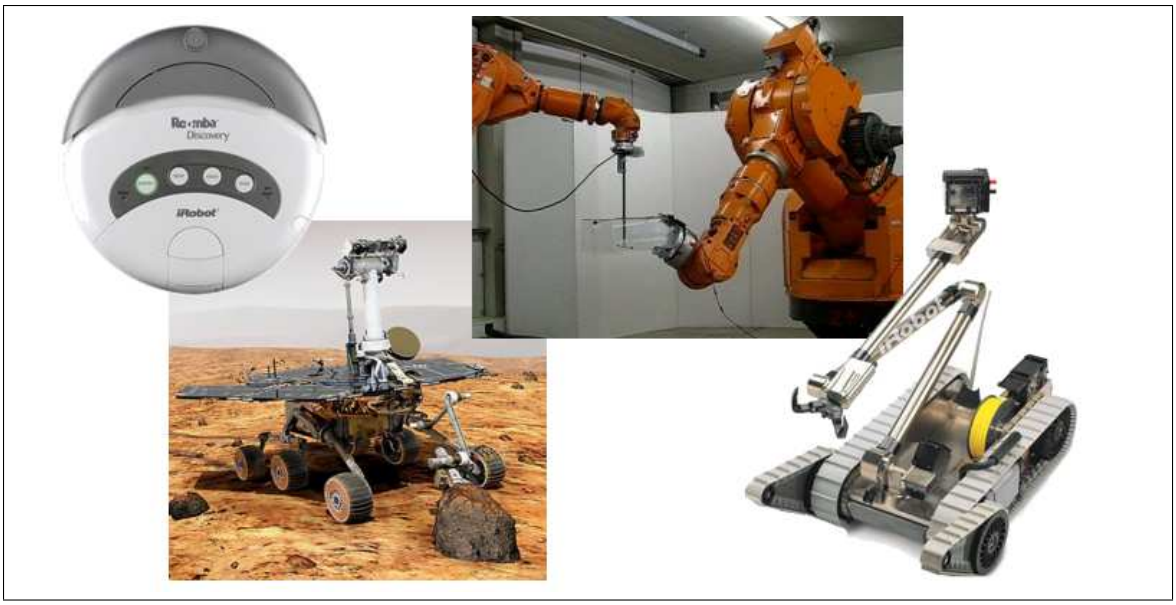


Figure 1.1. Some of the robots currently deployed in the real world. From left to right, Roomba Discovery for household cleaning, NASA Mars exploration rover, two Kuka industrial arms performing an assembly task and the iRobot Packbot EOD used for ordinance disposal

tasks in unconstrained, dynamic environments (Packbot, the Mars rovers, etc.), are generally tele-operated by a human operator. In all of these examples, the autonomy of the robot, if it exists at all, is extremely limited. Generally autonomy, the ability to accomplish an unconstrained set of tasks in unconstrained real world environments, is a capability that is significantly beyond the capabilities of these robots.

Figure 1.2 provides a graphical representation of this continuum. The horizontal axis is the generality of the robot, e.g. how many different tasks the robot can perform. The vertical axis is the autonomy of the robot, how independently can the robot perform its tasks. Industrial automation robots are located at the lower left of Figure 1.2, they are neither general nor autonomous. The Roomba is quite autonomous, but not very general. The Mars rover is general but not autonomous. The goal of general autonomy is to develop a robot that lies in the upper right corner of the graph.

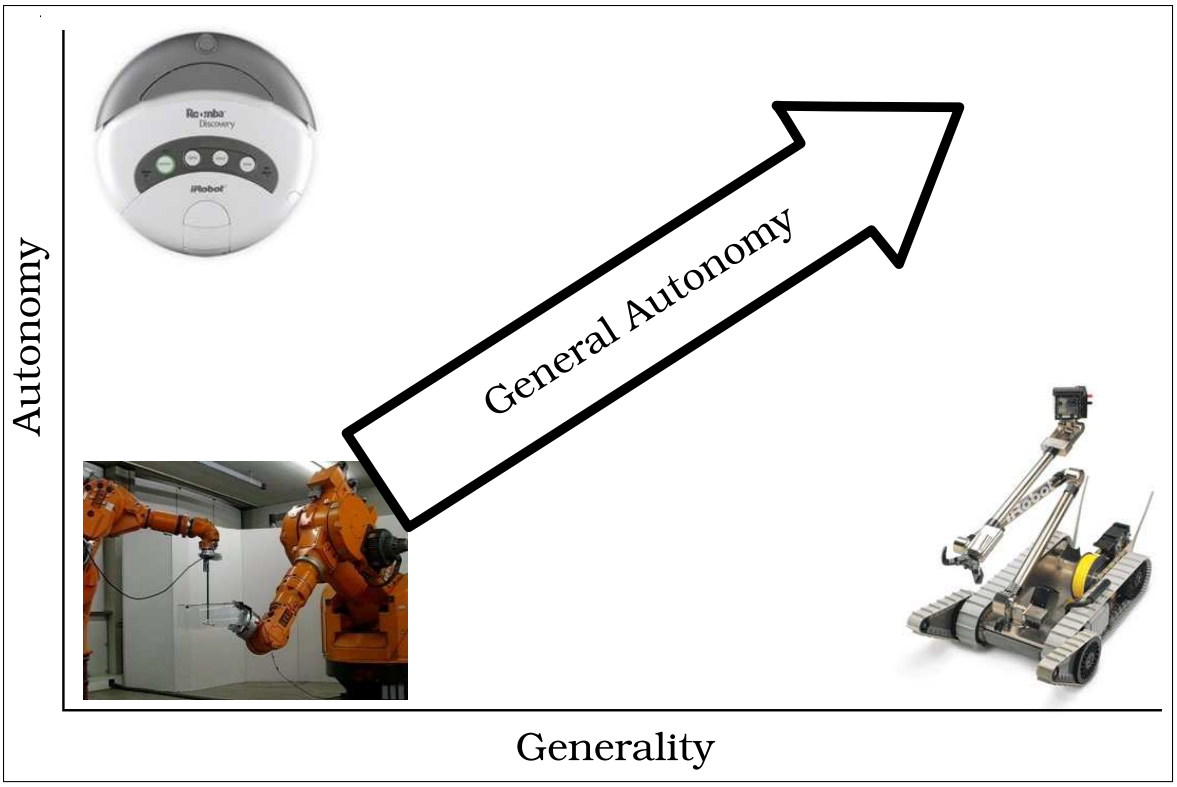


Figure 1.2. An illustration of current state of deployed robotics in the continuum of autonomy and generality as well as the direction leading to general autonomy.

Though there are many skills that are necessary for a robot to achieve general autonomy, one of the most significant problems that remains largely unsolved is the dexterous physical manipulation of objects in the world. Without the ability to interact with physical objects in a variety of sophisticated ways, robots can never be truly autonomous. Additionally, there are many practical robotic tasks, from elder care to building construction that require physical manipulation of the world.

Human beings manipulate their world in a myriad of sophisticated ways on a daily basis. On any given day, entering the computer science building and attending to tasks at my desk requires a large number of sophisticated manipulations of physical objects: I extracted my key card from my wallet. I swiped this card through the read to unlock the door. I grasped the door handle and opened the door. I opened three more doors in a similar fashion. I grasped my chair and moved it away from my

desk. I grasped a drawer handle and open the drawer. Once the drawer is open I may remove items or place items inside and then close the drawer. Human beings hardly think about such activities, we are so well endowed with manipulation skills that they are almost subconscious. Unfortunately, endowing a robot with similar skills is a significant challenge.

However, the rewards of solving this challenge are great. With such skills, robots would significantly increase their impact on our world. For example: Robots would become capable of dexterous labor and construction, massively reducing the cost of building structures from mundane office parks to human habitats on the moon and elsewhere. Robots with significant manipulation skills can also enable and extend independent lives for the elderly and the disabled. Providing these people with the ability to physically interact with the world even when their own bodies are physically incapable. Finally, dexterous robots could replace humans in hazardous situations requiring more sophisticated action such as fighting fires and mining.

The first requirement for such dexterous activity are arms whose morphology and capabilities match those of humans. Our physical world has been engineered with our morphology in mind. Countless everyday objects from door-knobs to pens to cars have been designed to be used by human hands and arms. Rather than re-engineer these objects, robots must have similar morphology. Beyond engineering of the physical world, the shape and flexibility of human arms enables our dexterous interactions with the world. Our arms have sufficient degrees of freedom to allow us to reach into complicated spaces such as inside the garbage disposal, or behind a book shelf. At the same time, these degrees of freedom are arranged so that there is also redundancy in our arms. This enables us to simultaneously achieve multiple tasks such as the rotation and upward force necessary to screw a light bulb into its socket. For a robot to successfully manipulate its world, it must have arms with physical characteristics similar to humans.

Of course, simply possessing the appropriate physical arms does little to enable a robot to manipulate its world. The robot must also be capable of moving these arms in an appropriate manner to satisfy the demands of the task being performed. Human beings are so good at achieving this that we simply think: "Please move my hand near that cup" and our body obeys. From a robot's perspective, the task is not nearly so simple. Robotic motion planning consists of two basic tasks. First, the robot must understand what poses are acceptable (e.g. collision-free) in a given environment. This requires classifying the environment as obstructed or collision free. Second, the robot must understand how to transition through a series of poses so as to transition from some starting position to some goal position. This requires understanding the connectivity of the free poses in the environment. Both of these tasks are components of the motion planning problem.

Generally these tasks have been tackled seperately. Local planning/control algorithms solve the task of maintaining the robot in a collision-free state. These methods are called "local" because they only consider robot's adjacent surroundings when determining how the robot should move. The task of determining connectivity is left to *global* planners which consider the robot's entire environment when computing a plan. While local algorithms are generally computationally efficient. Global planning requires significant computation.

In the general case, the global motion-planning problem is computationally intractable. The computational cost of exactly solving a motion planning problem is exponential in the number of degrees of freedom possessed by the robot [86]. The real-world is constantly in motion. This changing environment can place tight constraints on the time available for to a robot for planning. The computational complexity of the planning problem makes meeting these time demands quite challenging. Further complicating the problem, in real-world planning, knowledge about the environment is obtained from noisy sensors and is therefore inaccurate and incomplete. Consequently,

real-world motion planning algorithms must be able to compute successful motions not only efficiently, but also from incomplete and potentially inaccurate knowledge of the world.

Algorithms for addressing these challenges of robotic motion planning have existed almost as long as robots themselves [80]. In this history there is generally a distinction between local methods that act reactively to sensory input and global methods which construct and then execute complete plans of action.

Because local methods deal only with immediate sensory information they run extremely quickly, enabling them to respond to very fast changes in the environment. Unfortunately, this speed comes at the cost of completeness. Local methods generally greedily follow some objective function. Consequently the may become stuck in local minima of the function and are unable to proceed toward the final goal.

Global methods solve this problem by constructing a complete representation of the connectivity of the robot's *configuration space*. Briefly, the configuration space is the multi-dimensional space made up of all possible poses (or configurations) of the robot. The connectivity of this space is a function of configuration space obstacles, areas of configuration space that represent poses where the robot is in physical collision with itself or other objects in the environment. The configuration space is discussed in formal detail in Chapter 2.

There are many different algorithms for determining the connectivity of the configuration space. Some operate by examining the Voronoi diagram for the configuration space. Other algorithms decompose the configuration space into connected cells of free and obstructed space. Despite the additional computational complexity of examining the entire configuration space, when the dimensionality of the space is small and/or the structure of obstacles in the configuration space is easy to determine, these methods are efficient enough to handle configuration spaces that change as a result of the movement of obstacles in the real world.

Unfortunately, robotic arms with humanoid morphology make planning significantly more complicated. As previously mentioned, the computational complexity of motion planning is exponential in the degrees of freedom possessed by the robot. Even roughly approximating a single human arm requires seven degrees of freedom (three in the shoulder, one in the elbow, three in the wrist). Adding mobility adds an additional three dimensions. Though this flexibility significantly improves the tasks available to such robots, it also increase the complexity of motion planning to the point where complete understanding of the connectivity of the configuration space is computationally intractable. In large part this is because the shapes of obstacles in the high-dimensional configuration space are complex and difficult to understand. Consequently, the most successful algorithms for planning in such high-dimensional spaces use sampling to approximate the connectivity of configuration space. These algorithms sample and examine configurations and trajectories through the configuration space. The information from these samples is assembled into an approximation of the configuration space connectivity. This approximation only implicitly represents the actual shape of the configuration space obstacles and thus is able to plan efficiently.

Since the development of sampling based algorithms, the primary area of research in these algorithms has focused on strategies for selecting the samples from which the approximation of configuration space connectivity is constructed. Initially sampling occurred uniformly at random, then several heuristic approaches were suggested. Importantly, none of these methods received feedback from the planner *while* it was in the course of planning. Thus information obtained in the planning process was unavailable to guide subsequent sampling.

Each sample selected by the planner provides an observation of the state of the configuration space and its connectivity. These observations provide additional information that expands or refines the current view of the world. The planner continues

to solicit additional information about its environment until it can compute an adequate plan. Unfortunately, each query has a computational cost and in the context of real-world motion planning, the performance of the planner is a first-class concern. Consequently, an efficient planner must carefully choose its queries in an effort to obtain maximally useful information. To be successful, planning algorithms must use their current incomplete knowledge, as well as the task that they are solving, to guide the acquisition of useful new information. This thesis presents *utility-guided planning* a formal framework for structuring this guided exploration. This general framework advances the state of the art in robotic motion planning and has been successfully used to address two significant challenges in the development of motion planning algorithms suitable for real-world robotics.

1.1 Utility-Guided Planning

Utility-guided planning is an algorithmic framework for planning actions. Utility-guided planning is an iterative approach that begins with no knowledge of the world and incrementally constructs an approximation of the environment sufficient for planning. Every utility-guided planner is guided by a utility function that characterizes the expected relevance of a particular exploration to the planners progress toward solution. A utility-guided algorithm alternately selects such useful new explorations of its environment using its current knowledge and incorporates the resulting new information into its current approximation. In turn, the new approximation is used to guide subsequent exploration and the process of guided exploration continues until the plan is complete.

Because of the computational complexity of motion planning, the planner's knowledge of the configuration space is necessarily limited to a subset of the complete space. When knowledge of the world is limited, it is important to make maximal use of the knowledge that is available. Utility-guided planning accomplishes this by recognizing

that there is environmental structure in the space of potential movements available to a robot. Utility-guided planning uses this structure and its limited knowledge to construct an approximate model of the complete state of the configuration space. This approximate model provides predictions about those parts of the world whose actual state is unknown or uncertain.

In addition to predictions about the configuration space, an approximate model also suggests to the planner areas of the world that bear further examination. Examination of the approximate model indicates areas where the model is accurate and areas where the model is inaccurate. The planner can choose to refine its understanding of these inaccurate areas by acquiring further information. This process of model-guided refinement is known as *active learning* and has been studied extensively in context of machine learning.

However, there is a significant difference the goals of traditional active learning and those of motion planning. Traditional active learning is *model-guided*. Active learners acquire knowledge to improve the predictive accuracy of the approximate model they are constructing. The goal of an active learner is to maximize the accuracy of resulting approximate model. Active learners select explorations that maximally improve the accuracy of the model. In most cases they are attempting to learn a function that matches as closely as possible the function that generated the observed data. In contrast, the goal of a motion planner is an approximation of configuration space connectivity sufficient for computing some set of collision-free motions for a robot in an environment. The connectivity of configuration space is a relation rather than a function. Further, a complete understanding of this relation is not necessary to develop an approximation of configuration space connectivity sufficient for solving all motion planning queries. Consequently, the explorations selected by the planner are those that are most *useful* for computing this motion. The planner assess the *utility* of each exploration and selects the one that provides the greatest benefit to

the planner's understanding of its environment. The utility of each exploration is defined in the context of a particular task. This enables the general utility-guided framework to be applied to a wide variety of specific motion planning problems.

1.2 Contributions

Utility-guided motion planning advances the state of the art in robotic motion planning in several important ways.

The general framework for utility-guided planning represents an important conceptual shift in motion planning algorithms. Previously sampling-based motion planning has been a blind process. Whether random or heuristically guided, the samples that were selected were selected regardless of previous sampling or the planner's current representation of configuration space connectivity. In contrast the utility-guided framework is an online, adaptive approach to planning. Each time it chooses a sample to explore, it introspectively examines its state to determine the most useful sample to select. Information obtained from observing this sample is incorporated into the planner's representation of the environment and used to select subsequent samples to explore. The benefits of this utility-guided framework have been demonstrated in three different, but related motion planning problems: sampling configurations for probabilistic roadmap planning, guiding expansion of random-trees and guiding exploration in planning based on uncertain knowledge of the environment.

Theoretical analysis of sampling-based algorithms shows that their performance is dominated by the area of relevant configuration space that is least likely to be sampled by the planning algorithm. Consequently, the performance of a sampling-based planner is directly dependent on improving the probability of sampling relevant regions of the configuration space. This "narrow passage" problem [39] has been known long before this thesis. In the past decade, it has been one of the most heavily researched problems in sampling-based planning. Generally this research has proposed ad-hoc

and heuristic sampling strategies. These strategies are described in detail in Chapter 2. The utility-guided sampling strategy proposed in this thesis represents the first principled solution to the narrow-passage problem. Utility-guided sampling differs from these approaches in two important ways. First, it is the first sampling strategy that attempts to assess the expected impact of the sample prior to its exploration. Second, it is the first sampling strategy that is adaptive, incorporating information obtained from previous samples into the selection of subsequent explorations. As a result of these differences, the application of utility-guided planning to sampling significantly improved the performance of sampling-based planning.

Utility-guided planning has also been applied to random-tree planning. Random-tree planning approximates configuration space connectivity by expanding a configuration space tree rooted at specific configurations. Generally this approach is used as a *single-query* planner which finds a specific path between start and goal nodes. In random-tree planning there are numerous decisions that affect the manner in which the tree expands. Utility-guided planning provides a general framework for making all of these decisions. The location, direction and distance of the tree's expansion are all selected to maximize their usefulness in obtaining a solution to the particular query being addressed by the planner.

The utility-guided framework also enables motion planning using workspace information obtained from sensors. Previous work in motion planning assumes that information about the location and size objects in the workspace is perfectly accurate. However, when information about the world is obtained through noisy sensors, this assumption is necessarily false. Models of the world built from noisy data potentially contain substantial errors. Consequently, the information in these models is necessarily uncertain. The utility-guided framework is used to incorporate this uncertainty into the planning process. In this approach, the possible error in the environment is modeled and used to predict the actual state of the real world, given the

information observed from sensor data. These predictions are combined with a utility measure that provides the cost of failure (e.g. attempting a motion that is though to be free but in fact is in collision). This approach enables the planner to select paths that minimize the expected cost of enacting the motion. The utility-guided framework provides a general approach to planning under uncertainty that can be customized to different environments, physical robots and tasks using different utility functions. For example, a robot handling nuclear waste has a significantly different cost of failure than a robot handling a rubber ball. At times the expected cost of a path may be too high. In such situations, the robot must expend further energy sensing the world to reduce uncertainty.

Utility-guided planning also provides a formal algorithmic framework for this refinement. A robot's initial sensing of the world may be coarse. While this low fidelity representation may be adequate for some areas of the workspace, like large open areas. By examining its previous path planning, *utility-guided exploration* can identify those parts of the workspace that are already adequately known while also discovering areas of the world that require further sensing to develop a higher fidelity representation. This guided application of exploration enables a robot to successfully motion plan in the real world while expending significantly less energy sensing its environment.

A final contribution that is used throughout this work is the construction of predictive models of configuration space using statistical methods. These models enable the approximation of configuration space using a small sampled subset of the entire configuration space. This approximate model provides predictions about the state of unknown regions of the configuration space. Utility-guided planning uses these predictions to calculate the expected utility of an action prior to enacting it. To my knowledge, utility-guided planning is the first time such approximate models have been used in the context of the configuration space and motion planning.

Chapter 2
Background

This chapter provides background material, definitions and previous research relevant to this thesis. The problem of planning motions for robots has been examined nearly as long as robots have existed. However, much of recent work (this thesis included) relies on the classic formalization of Lozano-Pérez [71]. The following sections provide an overview of formalizations, terms and algorithms developed throughout the recent history of motion planning. There have been a number of books that provide a significantly more extensive examinations of the subject ([23, 59, 34, 62]).

2.1 Definitions

Robots

A *robot* is a physical object capable of controlling its pose in response to sensor information. Each robot is up from a collection of geometric parts whose collective volume define the robot's physical presence. For every robot there is a minimal set of parameters that uniquely specify each possible pose. A particular parametrization of the robot results in a particular orientation of its constituent parts. Each parametrization and its corresponding pose is known as a *configuration* of the robot. Robots also generally have physical constraints which limit the acceptable range of values for the parametrization. These constraints often also limit the movement of each of their parts relative to each other. The size of the parameter set specifies the number of degrees of freedom possessed by the robot. For example, in a three-dimensional environment a robot has six degrees of freedom: translation along three

axis and rotation along three axis. It is important to note that robot may not actually be able to control all of its degrees of freedom; the Roomba can not control movement along the vertical axis. For simplicity, when a robot can not control a particular degree of freedom it is ignored to reduce the size of the set of possible configurations.

The workspace

Every robot exists in a *workspace* that contains *obstacles* that physically conflict with some of the robot's possible poses. Although real-world workspaces are three-dimensional, in the case of some robots, such as mobile platforms, it is useful to abstract the workspace into a two-dimensional projection. The geometry of obstacles in the workspace can be represented in many different ways ([35] provides a survey). In most traditional motion planning, obstacles are assumed to be static. This significantly simplifies the planning problem.

The configuration space

The space of all possible configurations of a robot is known as the *configuration space*. Each configuration q in a configuration space is a unique parametrization of the robot. The dimensionality of a robot's configuration space is equal to its degrees of freedom. Thus, the volume of the configuration space is exponential in the robot's degrees of freedom. The range of each axis is defined by the robot's physical structure. The formalization of configuration space condenses the robot's physical structure and geometry down into a single point in a higher dimensional space. This simplifies the specification of the motion planning problem.

In the presence of workspace obstacles, the configuration space becomes a mixture of free and obstructed configurations. Individual configurations are *obstructed* if their corresponding physical pose is in conflict with one or more obstacles in the configuration space. Configurations may also be obstructed if the robot's physical pose is in conflict with itself. Configurations that are not obstructed are labeled *free*.

Each workspace obstacle has a corresponding *configuration space obstacle* that contains all of the configurations which are obstructed because of intersections with that workspace obstacle. The union of all configuration space obstacles is a subset of the configuration space known as $C_{\text{obstructed}}$, the complement of $C_{\text{obstructed}}$ is C_{free}. Thus, configuration space is a binary space. It is the union of the sets of free and obstructed configurations:

$$C = C_{\text{free}} \cup C_{\text{obstructed}}.$$

Configuration space paths

Given a particular configuration space, a *collision-free path* through the configuration space is a continuous function:

$$P[0, 1] \rightarrow q_i \in C_{\text{free}}$$

The origin of the path, $P(0)$ is designated as the *starting* configuration. The conclusion of the path, $P(1)$ is designated as the *goal* configuration.

2.2 Motion planning

Given these definitions, the collision-free motion planning problem can be specified as follows:

Given a robot R, a workspace W and two arbitrary configurations q_i and q_j in the corresponding configuration space C, compute a path P such that $P(0) = q_i$ and $P(1) = q_j$.

Classically, this problem has been known as the (piano) mover's problem [96]. It has been shown to be PSPACE-hard [86] through a reduction of Turing machine decidability to the movements of a three-dimensional robot. Multiple provably correct

algorithms for solving this problem have been proposed [96, 23]. All of these algorithms have at best exponential performance in the degrees of freedom of the robot for which planning is being performed. Despite this computational complexity, numerous practical algorithmic approaches have been developed. Generally speaking these approaches can be split into two categories depending on whether they explicitly or implicitly represent the structure of configuration space obstacles. Implicit representations of the configuration space are built by sampling-based planners which select a subset of the configuration space to use for planning. Results from statistics [101], show similar bounds for sampling-based motion planners (Section 2.2.2) that use uniform sampling. The following sections detail the particulars for several approaches to exact/explicit and sampling-based motion planning algorithms. One of the primary points of comparison between different algorithm's is their *completeness*. The notion of completeness represents the strength of the guarantee that the algorithm provides regarding its ability to compute a particular path. A maximally complete algorithm provides an absolute guarantee that it can find a collision-free path connecting any two configurations if such a path exists. For some algorithms this guarantee is weakened to *resolution-complete* or *probabilistically resolution-complete*. These lesser degrees of completeness are described alongside the corresponding algorithms.

2.2.1 Exact Motion Planning

Exact motion planning algorithms explore the entire configuration space and develop an explicit, complete representation of of $C_{\text{obstructed}}$ and C_{free}. Consequently, exact algorithms are *complete* algorithms. If a path exists it is guaranteed to be found. For practical reasons, some planners discretize their representation of $C_{\text{obstructed}}$ at some specific resolution. The resulting coarse representation is possibly incorrect. Such planners are *resolution complete*, they are guaranteed to find a path if it exists and does not require movement finer than their specific resolution.

2.2.2 Sampling-based motion planning

Rather than calculating an exact representation of the configuration space, sampling-based planners, like the probabilistic roadmaps (PRM) [51, 52, 82, 113] and rapidly-growing random trees (RRTs) [61, 56] construct an implicit representation of configuration space connectivity and the location of configuration space obstacles. Because they are stochastic, sampling-based algorithms are *probabilistically resolution complete*. The probability of finding a resolution-complete path can be made arbitrarily large by drawing a sufficient number of samples.

There are two different categories of sampling-based methods: single-query and multi-query. Single-query methods only compute a single path between a start configuration and a goal configuration. They gain efficiency by only developing an understanding of the configuration space connectivity that pertains to this particular query. Multi-query methods approximate the connectivity of the entire configuration space. While this is computationally more expensive, the resulting roadmap can be used to quickly compute a path between *any* two configurations. Multi-query methods are well suited to static environments where the expensive precomputation is worthwhile.

2.2.3 Planning under uncertainty

Any real-world planning must deal with uncertain perceptions of the world. In most cases, the state of the world is derived from sensors that are prone to noise and error. In addition, the underlying control algorithms that are responsible for the physical motion of a robot also introduce error in the execution of motion plans once they have been computed. To be successful in the real-world, planning algorithms must be capable of successful operation in the face of this uncertainty.

2.2.4 Guided sampling-based planners

Motion planners require an understanding of a high-dimensional configuration space to compute collision free movements. Sampling-based motion planning con-

structs an approximate understanding using limited exploration. Because the planner relies on limited exploration, it is important that each exploration provide maximal information about the connectivity of the configuration space. Consequently, selecting maximally informative explorations is the key to the performance of sampling-based motion planners. The task of guiding exploration in this way is complicated by the incomplete representation of the configuration space possessed by the planner. As a result, many of the methods proposed for guiding exploration are ad-hoc or heuristic. The following sections describe methods of guided exploration in both probabilistic roadmap (PRM) and random-tree motion planning.

2.3 Control Algorithms

Motion planning algorithms take a global perspective on robotic motion. They consider many possible configuration states of the robot when calculating a motion. In contrast, traditional control algorithms only consider information local to the planner's current state when calculating the robot's subsequent actions. By limiting the information that they consider, control algorithms are capable of running several orders of magnitude faster than motion planning algorithms.

Another important distinction between control algorithms and motion planners is the incorporation of feedback. Control algorithms constantly sense the world and respond, enabling them to handle dynamic and uncertain environments successfully. Only recently, has the necessity of sensory feedback entered into motion planning research.

However, the speed and resilience of control algorithms comes at a price. Because they only use local information, control algorithms are susceptible to problems with local minima. The algorithm may choose an optimal action based upon local information but this action may be sub-optimal given global information. This significantly limits the set of problems which local control can solve. The problem of local minima

is well known in the world of control algorithms and numerous approaches to solving this problem have been proposed. Such algorithms begin to hybridize aspects of control and planning. This merging of planning and control is discussed in much greater detail in the beginning of the next chapter.

Chapter 3
Related Work

3.1 Introduction

This thesis is related to a great deal of work in a number of different fields. It combines aspects of several different disciplines within robotics, additionally it is connected to developments in the fields of machine learning and utility theory. In the following paragraphs I will attempt to put this synthesis into perspective, subsequent sections will provide details of the specific related work.

Traditionally, motion planning and control have been separate fields within robotics. However, this historical distinction is at best arbitrary and at worst harmful to the development of practically successful algorithms for generating robotic motion. It is more useful to see planning and control as existing on the same continuum. At one end is purely reactive control, capable of running thousands of times per second but only using the simplest local information to make its decisions. At the other is exact planning, requiring tens to hundreds of seconds to run, but using all available information. Neither end of this continuum are practical algorithms for generating real-world robotic motions. Reactive control is too simple and susceptible to local minima to be useful, while exact planning is too slow to react to even slowly changing environments. Practical algorithms, lie somewhere between these two extremes. For example, global potential field control methods [55, 26, 16] implicitly provide a plan for reaching a goal location from any other configuration. While other methods [17, 114] take motion plans and make them at least partially reactive to dynamic environments.

This thesis also introduces motion planning methods which push down the continuum toward control. My work improves the efficiency of sampling-based motion planning enabling it to run repeatedly inside a control loop and react to changing environments. The utility-guided approach to planning under uncertainty, explicitly incorporates high level sensor feedback and adapts its planning to react to new information. Because this thesis is primarily concerned with motion planning, I discuss previous approaches to motion planning, especially sampling-based motion planning in significant detail. Control algorithms are less directly related and thus discussed at a higher level and in less detail than motion planning.

This thesis views motion planning as a search for the set of actions which satisfy a particular path query. In any search problem, efficiency is achieved by asking the minimal set of questions required to find what you are looking for. The field of *active learning* within machine learning, has examined many ways to optimize the search for information. The planners in this thesis also use machine learning to generalize limited information about the configuration space into a broader approximation of its structure.

The key to efficiency when searching for a plan, is identifying explorations with maximal value. In active machine learning, this value is directly related to the reduction in model error provided by the exploration. Unfortunately, in motion planning, estimating the value of an exploration is significantly more complicated. To formalize these estimates I use *utility-theory*. Chapter 4 provides a detailed discussion of utility-theory and the general utility-guided planning framework.

3.2 Planning for robots

There have been far too many algorithms proposed for computing robotic motions to accurately summarize in the context of this work. In the following I present those works that bear most relevance to the work developed in this thesis.

3.2.1 Exact Planning

Exact planning algorithms examine the complete configuration space of the robot. Consequently, they are guaranteed to find a path if it exists. Unfortunately, this also means that they are subject to the PSPACE-hard complexity of motion planning. Practically, this means such algorithms can only be used for robots with a few degrees of freedom.

Roadmap Planners

A configuration space roadmap is a graph that represents the connectivity of the configuration space. For kinematic motion planning, the nodes in the graph are individual configurations in C_{free}, edges in the graph indicate a simple path through configuration space connecting configurations.

There are many types of roadmaps and methods for computing them. One of the earliest was proposed by Nilsson [80] for mobile robot motion planning. The configuration-space roadmap for general robotic motion planning was popularized by Lozano-Perez [71], who also proposed the first roadmap algorithm applicable to any configuration space. The most efficient exact roadmap construction algorithm is due to Canny [23] who reduced the complexity of Lozano-Perez's earlier algorithm from double exponential to single exponential in the degrees of freedom of the robot. Canny's Silhouette method projected configuration space obstacles into lower dimensional representations and traced their projections until a roadmap was computed. Canny's planner relies on mathematical results from semi-algebraic sets and is purely theoretical. His algorithm is both formidably complex and too slow to have practical application. A modified version of the algorithm capable of path planning in practice was later implemented [24], but was computationally limited to three or fewer degrees of freedom.

Cell Decomposition Planners

Rather than construct a roadmap of configuration space, cell decomposition planners break the configuration space into adjacent geometric cells. In exact cell-decomposition, these cells are labeled either "free" or "obstructed." The set of free cells *exactly* tiles C_{free}. Likewise the set of obstructed cells, exactly tiles $C_{\text{obstructed}}$. Given an exact decomposition of the configuration space, motion planning is achieved by moving from start to goal through adjacent free cells.

The primary challenge of exact cell-decomposition is determining the boundary planes of free and obstructed cells. While this is relatively feasible given rigid polygonal robots in a workspace with polygonal obstacles. It is generally intractable for articulated robots and non-polygonal workspaces. Exact cell-decomposition is impractical for situations in which a clean polygonal configuration space does not exist.

Rather than computing cells that exactly tile C_{free} and $C_{\text{obstructed}}$, approximate cell decomposition [18] tiles the entire configuration space with regularly shaped cells, for example hyper-cubes. Regular cell shapes require the addition of a "mixed" cell label. Cells that are mixed contain both obstructed and free configurations. In approximate cell decomposition, cells that are entirely free or obstructed are retained by the planner, while cells that are "mixed" are subdivided until a minimum cell size is reached. Like exact cell decomposition, motion planning is attempted through adjacent free cells. If a path through adjacent free cells is impossible, but motion through a mixed cell might allow a path, a local control method (Section 3.2.2) is sometimes used in an attempt to move through the mixed region. Because it subdivides configuration space, approximate cell decomposition is only feasible for relatively low dimensional configuration spaces. Like exact decomposition the computation of whether a cell is obstructed, free or mixed can be quite costly for configuration spaces whose structure is unknown.

3.2.2 Local Planning and Control

In contrast to complete planners that construct a global representation of configuration space, local methods of planning use only information local to the robot to make rapid control decisions.

Local planners impose a artificial *potential field* [53] function on top of the configuration space. This potential field function is sloped so that its minimum is at the goal configuration. The artificial potential field is also influenced by configuration space obstacles. Configuration space obstacles have high artificial potentials that decline gradually with distance from the obstacle. At any instance, the robot calculates the derivative of the potential function and descends the maximal downward gradient in an effort to reach the minimum at the goal position. This calculation quickly determines the motion to take next. Unfortunately, this approach to control causes potential field methods to get stuck in local minima where they are unable to proceed up an increasing potential function to reach the goal configuration. Additionally, while artificial potential field functions are easy to define for low dimensional configuration spaces that closely match the workspace, defining a potential function for a higher dimensional configuration space, or one where the location of configuration space obstacles is unknown is quite challenging.

To overcome these challenges, numerous methods have been proposed which compile offline plans into reactive policies. Examples of this include the navigation function NF1 [55] and harmonic potential functions [26].

Another important offline approach to compiling a plan into a control policy is reinforcement learning [102]. In reinforcement learning an offline simulated agent performs actions and receives a reward for specific outcomes. There are numerous different algorithms for learning policies from experience. Many of the learning algorithms are proven to converge on the optimal policy given sufficient experience.

Rather than compiling a plan into a potential function, *Receding horizon control*[57] explicitly plans a control sequence out to the horizon at time N. However, only the first command in the control sequence is executed. After the command is executed, the new state of the robot is observed and a new optimal control sequence is computed, this time from the current observation out to a horizon at time N+1. This approach has the benefit of harnessing a model to predict a series of actions while still being reactive to failures and other unexpected events.

Rather than pre-computing a plan and compiling it into a potential function, another option is to simultaneously control the agent while using information obtained from this activity to improve the control policy. Real-time Dynamic Programming [12], adapts offline dynamic programming algorithms to online operation. This has the desirable benefit of adapting learning to the areas required by the actual operation of the system without requiring a developer to (possibly incorrectly) predict the location of these state-space regions. This notion is closely related to utility-guided exploration.

Real-time dynamic programming uses a Markov decision process (MDP) to model the underlying behavior of the system. It also assumes that the structure of this model is known. Adaptive real-time dynamic programming is an extension of this approach to situations where the structure of the model is unknown. In addition to updating the control policy after each subsequent action, the state transition probabilities of the Markov model used for control decisions are updated as well.

Another approach [85] combines reinforcement learning with local control. Reinforcement learning is used to develop a policy which guides a two-link robotic arm into the domain of the local controller. This approach improves the feasibility of using reinforcement learning for online control by dramatically reducing the number of trials required to learn a successful control policy.

```
Probabilistic Roadmap Construction: Roadmap
        G = empty roadmap
        foriterations do
            q = configuration chosen uniformly at random
            if(q ∈ C_free)
              add q to G
              foreach q' ∈ Neighbors(n, G)
                  if(StraightPathPossible(q, q'))
                    add an edge from q to q' to G
                    break
        return G
```

Figure 3.1. The PRM algorithm

3.2.3 Multi-query sampling-based planning

The PRM algorithm is the most well known multi-query method. It constructs a roadmap graph by uniformly sampling configurations. If a configuration is free, it is added into the roadmap. When a configuration is added to the roadmap an attempt is made to connect it to nodes in the existing roadmap graph. For some free configuration q_i, the k nearest configurations in the roadmap are identified. Paths between the new configuration and its nearest neighbors are attempted using a simple straight-line planner. If the path is possible, an edge connecting those configurations is added to the graph. Given a probabilistic roadmap and pair of configurations, a path is computed by moving from each of the configurations onto the roadmap and then traversing the set of known free edges contained in the roadmap. The PRM algorithm is given in Figure 3.1.

Sampling-based planning has also been applied to cell-decomposition planning. The probabilistic cell decomposition planner [70] does not decompose the entire configuration space. Instead it uses a lazy approach, only subdividing the configuration space as necessary in response to specific queries. In this approach, cells are either "possibly free" (e.g. containing only free samples) or "possibly obstructed" (e.g. containing only obstructed samples). Whenever a cell is found to be mixed (containing

both obstructed and free samples) it is subdivided. If a path through a series of free cells can not be found, the algorithm samples the obstructed cells until a free configuration is observed requiring a new split.

Guided sampling strategies

As discussed in Section 2.2.4, guided sampling is the key to efficiency for practical motion planning in challenging environments. There have been many extensions to the uniformly random sampling strategy used by the initial PRM algorithm [52]. Generally, these extensions improve performance by reducing the number of samples required to construct a complete configuration space roadmap.

The Gaussian sampling strategy [15] and the bridge test [38] select configurations that are thought to be close to obstacles or inside narrow passages, respectively. Other heuristic sampling strategies modify obstructed configurations to discover nearby free configurations. These heuristic samplers use obstacle surface properties [1] or dilating and contracting obstacles [39] to modify colliding samples into free ones. All of these strategies are based on uniform random sampling and require additional computational effort to filter configurations to find those thought heuristically to be valuable. Despite this extra computation, only a subset of configurations selected by the heuristic are truly relevant to roadmap construction. All of these sampling strategies are filters on top of uniform random sampling. Thus, they do not reduce the expected number of samples. Instead, they only perform computation on those samples that are heuristically deemed worthwhile. This reduces the average computation per sample and thus the overall runtime of the planner.

An alternative obstacle-based approach modifies the size of obstacles to identify important regions to sample. The small-step retraction sampling strategy [92] shrinks the size of the geometric objects representing the physical robot. This has the implicit effect of expanding the available free space. This makes the planning problem easier

for a planner to solve. Subsequently the configuration space is shrunk back to its actual size. This may invalidate some of the paths found in the expanded free space. Attempts are made to repair these paths. The location of these repairs is used to guide subsequent sampling.

Visibility-based PRM planners [99, 60] label configurations that act as "guards." Such configurations capture a region of configuration space containing every configuration that has a straight line path to the guard. Only configurations that are not in a captured region, or connect two guards are inserted into the roadmap. The roadmaps that are constructed are significantly smaller, but the visibility region is quite expensive to compute.

Two approaches calculate and use the medial axis to minimize the probability that a configuration is obstructed [67, 36, 115], thus increasing the probability that a relevant sample is chosen and reducing the second term in the expected samples expression. A significant challenge to these approaches is the difficulty of finding configurations near the medial axis for articulated robots.

Several other guided sampling strategies use information obtained from previous experience to guide their behavior. Entropy-guided [19] sampling adapts sampling to find configurations that offer maximal information gain given the current state of the planner. The measure of information gain focuses sampling on regions not explored by the planner. However, the planner does not attempt to limit sampling to free configuration space, leading to potentially pathologic behavior.

The model-guided [21] sampling strategy chooses configurations that maximize the decrease in variance of an approximate model of configurations space. While these configurations are relevant to building a model with maximum accuracy they are not necessarily relevant to successful motion planning.

Hsu et al. [37] propose sampling using an adaptive mixture of sampling strategies. The mixture of strategies is adapted based on the past success of each strategy.

Workspace information can also be used to guide sampling. Yang and Brock [116] identify narrow passages in workspace obstacles and use these workspace locations to select configuration space samples.

Another way to guide exploration is to defer exploration until it is necessary to solve some specific query. The LazyPRM [14] algorithm defers all collision-checking of edges and nodes in the roadmap until a particular path query is made. In response to this query, LazyPRM verifies edges that are thought to be relevant to the query. If the planner is unable to find a collision free path, information from invalid edges is used to guide subsequent sampling of the configuration space.

Another method of deferring exploration is to adjust the detail of the exploration. FuzzyPRM [78] estimates the probability that each edge in the roadmap is obstructed or free. Edges are selected for refinement based upon their probability of being free. Refinement either invalidates the edge or increases the probability that it is free until it is certain.

Exploration can also be guided to accomplish a specific task. Yu and Gupta [118], also use entropy and information theory to guide exploration. In their work, it is used to maximize the quality of the observations of the world obtained by a eye-in-hand robot.

The value of randomness in sampling has been questioned by some research. In particular, random sampling makes it difficult to control the dispersion and density of samples of the configuration space. A proposed alternative [63] is to sample on a quasi-random lattice using either the Hamilton or Hammersley sequence. This ensures that sampling obeys the desired dispersion and density. These quasi-random sampling strategy has been used for both multi-query and single-query planning.

3.2.4 Single-query sampling-based planning

Instead of constructing a complete roadmap, capable of responding to multiple queries, single-query planners achieve efficiency by focusing their search on a specific path. The most well known of these approaches are *random-tree planners*. Random tree planners grow one or more configuration-space trees rooted at specific configurations. Generally, these planners operate by rooting a random tree at the start and goal configurations and expanding these trees until they meet and a path can be computed. The expansion of these trees is governed by three choices: the node in the tree to expand, the direction of expansion and the distance of exploration.

One of the first algorithms for random-tree planning is the Z^3 [10]. The algorithm grows a tree from start to goal through a series of randomly selected sub-goals. Paths connecting nodes in the tree are computed using a local planner that attempts local obstacle avoidance. A parallel version of the same algorithm [11] improved the efficiency of the planner.

Ariadne's Clew [72] also performs random-tree expansion by simultaneously solving two optimization problems. The first performs a local search to find paths connecting a pair of points. Because this search is local it may become stuck in local minima. The second optimization problem distributes landmarks in the configuration space. These landmarks are connected to form a complete path using the first local planner.

The most widely used tree-based planners are the rapidly-growing random tree (RRT) family. The original RRT-Connect [64] algorithm simultaneously selects a node for expansion and a direction of expansion using the Voronoi bias that directs expansion toward large unexplored regions of configuration space.

In RRT planning, one of the primary considerations is the trade-off between exploration and refinement. One way to emphasize exploration is to reduce the dispersion of random-tree expansion. Dispersion-RRT [68] modifies the RRT algorithm to select

a set of random samples rather than a single sample. These samples are ordered by their distance to their nearest neighbor and then expansions are attempted in order by the pair (sample and nearest neighbor) that are farthest apart. This approach emphasizes long distance paths through configuration space maximizing exploration while minimizing refinement.

A more adaptive approach to balancing exploration and refinement is to adaptively restrict the nodes that are considered for long distance expansion. Dynamic-Domain RRT [117] and Adaptive Dynamic-Domain RRT [46] limit the length of node expansion in the proximity of obstacles. Once a node and an expansion direction have been selected, all of these methods use a static length for the expansion.

Like most sampling-based planners, basic RRT does not consider path cost when expanding the tree. Urmson and Simmons [106] propose an extension that does consider a path cost function to bias the selection of expansion node and direction. The cost function used is the ratio of the current path to the optimal straight-line path through configuration space. This work also suggests expanding the k-nearest neighbors or the k-best nearest neighbors rather than simply expanding a single node. The quality of a node for the k-best selection is heuristically estimated with a variety of different functions including proximity to obstacles.

OBRRT [87] uses the Voronoi bias to select nodes for expansion, but uses a hybrid approach for expansion direction and distance. OBRRT uses weighted random selection to choose between six different methods of selecting an expansion method. Once an exploratory expansion has occurred all four approaches attempt to connect the newly added node to the closest node in the other configuration space tree.

The Blossom-RRT [49] algorithm is a refinement intended only for motion planning with a small discrete set of possible actions available to the robot. Rather than selecting a single expansion for a particular node in the random tree, Blossom-RRT

explores all possible expansions of the node. As a result, each node is only ever expanded once. Once expanded, it need never be visited again.

The RRFT algorithm [54] modifies RRT planning to test reactive control systems. RRFT proposes two methods for selecting nodes for expansion, one based on system dynamics and one based on a history based weighting. It also proposes an adaptive bias of expansion direction toward the goal region of the state space.

The expansive space approach to tree-based planning [40] selects nodes for expansion based upon the density of nodes in the configuration space. Nodes that are in sparsely sampled areas are more likely to be selected. Rather than a single expansion, the selected node is expanded in multiple directions. These directions are selected with a bias toward directions that lead to areas of the configuration space sparsely covered by the tree. The distance of expansion must be less than a neighborhood threshold. After each expansion, connection is attempted between every pair of nodes in the two trees whose distance is less than a threshold. The expansive space approach has also been extended [83] to use a node selection function that incorporates a heuristic cost function and the degree of the node in addition to tree density to bias node selection.

The adaptive framework for single-query planning [107] is a hybrid approach to tree-based planning. This approach focuses solely on connecting the trees together. Only failed attempts at connection are used to expand the trees. In each iteration, a pair of nodes, one in each tree, and a path planning algorithm for connecting them is selected. These nodes and algorithm are selected using a heuristic scoring function that weighs local configuration space properties of the start and goal nodes, global properties of the space between the configurations and planner characteristics.

The SBL [93] (Single-query, Bi-directional, Lazy) algorithm is another random-tree based algorithm. Nodes are selected for expansion in proportion to the density of nodes in the surrounding region of configuration space. Once a node is selected,

expansion occurs by sampling from iteratively smaller hyper-spheres surrounding the node until a collision-free configuration is found. Once such a configuration is found, it is immediately added to the tree. Verification of the connecting trajectory is not performed. This improves efficiency, but introduces the possibility that the resulting path may be invalid.

The SRT [81] (Sensor-based Random Tree) algorithm, extends random tree planning to environments where obstacles are unknown and must be discovered by the robot. This approach augments the traditional configuration space roadmap with "local safe regions" which represent the free-space surrounding particular nodes in the roadmap. When selecting directions for expansion, the algorithm selects configurations which are within the safe region of the current configuration but not in the safe region of any other configuration, this favors expansion and exploration. SRT has only been applied to 2D mobile robots. Identifying the safe-regions via sensing relies on the easy mapping between workspace and configuration space available in 2D mobile robotics. Consequently, extending the approach to jointed arms and other robots with many degrees-of-freedom would be very difficult.

3.2.5 Hybrid Planners

In addition to strictly single or multi-query methods, there are planners that hybridize multiple different approaches to motion planning. These methods observe that different techniques may be better suited to particular regions of the same configuration space. Hybrid methods attempt to apply the appropriate technique to the appropriate region of configuration space and combine each planner's result into a single final plan. For example, LazyPRM [14] was originally proposed as a single-query planner that can also be used to solve multiple sequential queries.

Morales et al. [76] propose a method that splits configuration space into regions and selects different planners depending on the features of the regions. The selection

is guided by a decision tree that is learned offline. While the selection of planners is guided by observation of configuration space, the methods themselves are not adaptive.

Plaku et al. [84] describe a planner that hybridizes traditional multi-query PRM and RRT planning. The planner constructs a number of disjoint multi-query graphs in the configuration space and then uses single-query planning in an effort to connect the various disjoint graphs. An additional feature of this decoupled approach is that it is easy to parallelize the algorithm to run simultaneously on multiple processors.

Isto and Saha [44] describe a hybrid planner that for very high dimensional configuration spaces. The planner achieves tractable performance by connecting samples to the roadmap via paths computed on a lower dimensional sub-manifold of the original problem. Sub-manifolds are either sampled randomly or selected using a priori workspace information. The lower dimensionality of the sub-manifolds enables the efficient application of existing motion planning methods.

To be capable of real-world motion planning, planners must also be able to deal with obstacles that move. Van Den Berg et al. [109] propose an approach that performs motion planning in the configuration-time space which encompasses both configurations and time. A two-level search of configuration-time space is used to compute a path. A lower level local planner identifies successful local trajectories. The A* planner is used to combine these local trajectories into a complete global plan.

3.2.6 Planning with uncertainty

Nearly all traditional planning methods both complete and local assume that their perception of the world is accurate. Several methods of local control have been proposed for handling sensor error. The most common approaches are the Kalman filter [50] and its approximation, the particle filter [7]. These approaches model the expected error in sensor observations as well as the expected change in the sensor

value. This information is used to construct a distribution representing the true state of the world given the noisy sensor values.

A significant amount of recent research on uncertainty in robotics has focused on the problem of simultaneous localization and mapping (SLAM) for mobile robots [104]. The work in SLAM provides some of the inspiration for the use of modeling and probability estimation in order to minimize uncertainty. SLAM solves an important problem, namely how a maximally accurate model of the workspace can be constructed from noisy and erroneous sensor data. A workspace model, however, is not sufficient for motion planning. For motion planning, the uncertainty in the model of the workspace must be translated into uncertainty in the model of the configuration space. In turn, this uncertainty must be integrated into the planner to identify paths that minimize the expected cost, due to failure, of the motion plan. Thus SLAM is a starting point upon which this approach builds, the maps of the world produced by SLAM algorithms (as well as any estimates of the map's certainty) are a possible source for the representation of the workspace used to compute a motion plan.

There has been generally more work on the uncertain execution of plans. Traditional local control algorithms respond to execution error by constantly reassessing the error in the system and taking appropriate action to correct this error. Many such algorithms can be mathematically proven to converge (in the limit) on the desired state. Reinforcement learning [102] achieves a similar outcome at a higher level. Reinforcement learners develop a *policy* for action given the state of the agent. Because this policy applies to many different agent states, an agent using such a policy is partially robust to stochastic execution of its actions. Regardless of the state in which the planner finds itself after taking some action, the policy provides a subsequent action to take. Many reinforcement learning algorithms also converges on the optimal policy (given some reward function) after enough experience.

Kaelbling et al. [48] model the world as a partially observable Markov decision process (POMDP). The state space for the POMDP is a set of abstract observations of the environment (door, hallway, etc) and abstract actions (move-forward, turn-left, etc). This abstract state space is constructed on top of a lower level system which is error prone. The POMDP formulation explicitly represents and reasons about the possibility of sensor error. Several heuristic strategies for approximating the computationally intractable optimal policy are suggested. These approaches have been successfully used for mobile robot navigation in office environments. The POMDP formulation requires that the state space of the robot be discrete or that it be represented by a discrete approximation.

Motion planing in uncertain real-world scenarios often requires feedback from the environment. The area of feedback motion planning has recently received some attention in the literature. Conner et al. [30] propose a hybrid control system which combines a set of local control policies with a discrete planner to develop a planner for mobile robots in obstructed environments. Lindemann and LaValle [90] propose a general theoretical approach to feedback planning using cylindrical algebraic decompositions of the configuration space. The elastic roadmap [114] consists of a series of configuration space milestones and local controllers which link the milestones together. Plans computed in the roadmap use the local controllers to maintain task constraints and obstacle avoidance while executing the plan. van den Berg and Overmars [109] also propose integrating a roadmap and local control. Their planner, which is intended to handle dynamic obstacles, computes a roadmap for static obstacles in the environment and then uses a local obstacle avoiding controller to execute the edges that connect two milestones together. In contrast to this work, none of these approaches explicitly consider integrating uncertainty or sensory refinement and planning.

The problem of sampling-based motion planning in dynamic environments was first considered by Leven and Hutchison [65] who propose a roadmap representation that can be efficiently modified as obstacles move. Another approach [45] assumes a finite set of dynamic obstacles, such as doors, whose motion is known. In this work the roadmap is augmented to label edges as possibly obstructed by a dynamic obstacle. When a path is computed in the roadmap, these possibly obstructed edges are re-checked to ensure they are currently free. If a path can not be found, a local planner is used to reconnect and replace the obstructed edge in the roadmap. A further refinement of this hybrid roadmap approach [108] also constructs a roadmap for the static portions of the environment. When the roadmap is queried it is augmented with a time dimension to enable consideration of dynamic obstacles and changing environmental properties. They also propose incremental re-planning of the path as new information is obtained in the process of plan execution. However, they do not explicitly guide acquisition of new information.

Missiuro and Roy [74] address sensor uncertainty in the context of complete motion planning. The approach adaptively samples configuration space using a distribution based upon the certainty of the sampled configurations. Paths are found in the resulting roadmap using A* search and an uncertainty heuristic. This work uses a vertex based model of uncertainty and has been applied to a 2-DOF mobile robot.

There has also been work on guided exploration for a motion-planning system. Yu and Gupta [118], use entropy and information theory to guide exploration to maximize the quality of the observations of the world obtained by a eye-in-hand robot. In this work the purpose of the exploration is to minimize entropy in the representation of the entire workspace, not reduce the uncertainty of a particular motion plan.

3.2.7 Non-Traditional Applications of Motion Planning

Motion planning algorithms have also been applied to an array of problems beyond traditional robotics. The two main areas of application are computational biology and computer games and animation. Though not the main focus of this thesis, these non-traditional applications of planning may benefit from many of the improvements in performance and reliability offered by the utility-guided framework.

The final shape or *conformation* of a protein indicates its function. Unfortunately, determining the shape of a protein is a currently a complicated and expensive process for traditional wet biologists. Recently researchers [3, 4, 2, 5] have proposed a computational alternative. This research views the protein as a flexible robot with many degrees of freedom. Rather than discovering connectivity in the binary configuration space, motion planning occurs in a continuous energy landscape or *conformation space*. However the same motion planning algorithms are used. Motion planning techniques have also been used to simulate the motion of flexible molecules [29].

Many computer games are populated by numerous agents controlled by the computer. For example, the elements of an opposing army or the denizens of a virtual city. The coordination of the movement of all of these agents is quite difficult. Often the game simply generates movements for each of the agents individually. Unfortunately this significantly impacts the realism of the result movement. Often it impacts the quality of the game as well. Without coordination, one member of an army unit may choose to move in a manner different from its cohort. As a result, that unit is easily destroyed. To resolve this problem, Niewenhuisen et al [79] propose viewing the set of individuals as a single robot with many different degrees of freedom that control all of the individual units. Viewed in this way, traditional motion planning algorithms can provide movements for all of the individuals that coordinate the movement of the whole unit also.

Virtual characters, such as those in computer generated animation and virtual environments also often have many degrees of freedom to increase their realism. Traditionally, animating the motion of such characters requires a human animator to manually input a large series of poses. However, such virtual characters can also be viewed as robots. Consequently, several researchers [6] have used motion planning techniques to generate character movements. This application of sampling-based techniques reduces the burden on the human animator and enables significantly more flexibility and accuracy in the generated motions.

Sampling-based motion planning has also been applied to the task of untangling mathematical knots [58] a problem known to be NP-Complete.

3.3 Active Learning

Configuration space can be viewed as a binary classification, $C(q) = 0, 1$, which takes some configuration q and returns whether or not that configuration is obstructed. Because of the topological properties of configuration space, if some q is obstructed, it is more likely that its neighbors are also obstructed. The same is true of free configurations. Given a collection of sampled configurations that have been labeled with their state, we can use learning methods from machine learning [75] to construct an approximation of the function C which is I call $\tilde{C}$. This approximation function returns a number between zero and one, estimating the likelihood that a particular configuration is obstructed or free. This approximation function is the approximate model of configuration space. There are many different algorithms for constructing this approximate function. Section 5.4 discusses the ones used in the context of this thesis.

Of particular relevance to this thesis is the sub-field of machine learning that deals with *active learning*. The term "active learning" encompasses a variety of techniques in machine learning that use the state of their current classification model

to select training examples expected to maximize the resultant improvement in the model. There is a significant connection between active learning and sampling-based motion planning. In both cases, the algorithms have the opportunity to incrementally select the examples from which they acquire information. Also, both sampling-based planners and active learners improve efficiency by minimizing the number of examples required to develop sufficient information for their task (either modeling or planning).

In active learning, many methods for selecting explorations have been proposed including areas where there is currently no data, areas where there is little certainty about the model [103, 66] and areas where the model has low accuracy [69]. The most significant work either focuses on reducing the size of the version space, or reducing statistical error.

The *query-by-committee* [33] algorithm selects areas for exploration on which different classifiers (the committee) disagree. The query-by-committee algorithm was combined with expectation maximization (EM) by McCallum and Nigam [73]. In the "expectation" phase, the current classifier was used to guess labels for unlabeled data. In the "maximization" phase, these guessed labels are used to further train the classifier. Their approach also proposed using the Kullback-Liebler divergence as a metric for measuring committee disagreement. *Vote entropy* [31], is another measure of committee disagreement used for query-by-committee active learning. Vote entropy measures the entropy of the distribution where each committee member in a group of size k contributes a probability mass $1/k$ to its preferred class.

Tong and Koller [105] present a similar approach which selects examples which maximally reduce the size of the version space of a support vector machine (SVM) [110, 111, 100, 94]. SVMs project training data into a much larger feature space through kernel functions. Because this decreases the density of the space, it is much easier to classify the projected data using hyperplanes.

In contrast to these ad-hoc approaches, Cohn, Gharamani and Jordan [25] show that a learner which selects configurations that maximize the reduction in variance of the resulting model construct optimally accurate classifiers for the number of samples examined. Unfortunately, their approach is only applicable to problems where a closed form expression for the error is available. To resolve this, Roy and McCallum [88], use Monte Carlo estimation of the reduction in error afforded by each potential example to select one which maximizes the reduction in error. This method is both principled and can be practically applied to a variety of different problem domains.

Experimental evidence [91] shows that models built using active learning guided by model variance significantly outperform models constructed by uniform random sampling in most domains where different examples carry varying information (as is the case in configuration spaces).

These results provide inspiration and support for the benefits of utility-guided planning. An important difference is that while active learning methods are guided by the variance of the approximate model being learned, utility-guided planning attempts to optimize some additional *utility function* related to the task of the planner rather than the learned model.

Chapter 4
Utility-Guided Planning

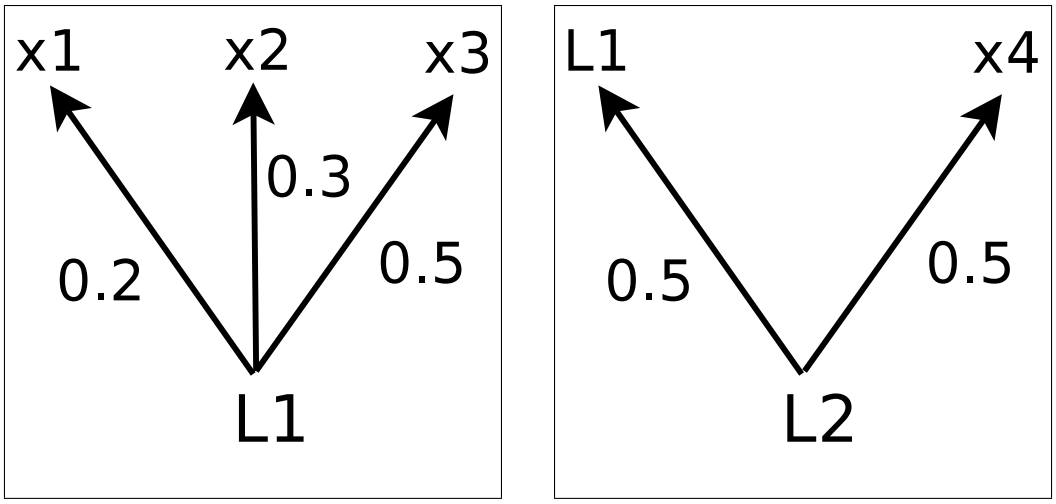
Utility theory has its beginnings in the field of economics. It was developed to gauge peoples seemingly non-linear valuations of assets under different risk conditions. However, since its introduction it has been successfully applied to many tasks beyond economics. The following first gives an overview of general utility theory, then describes its relevance to motion planning and finally provides an outline of its specific application to multi-query and single-query planning as well as planning which considers uncertainty.

4.1 Utility Theory

Expected utility presents a formal approach to specifying and evaluating an agent's preferences regarding actions with non-deterministic outcomes. In utility theory, these actions are referred to as *lotteries*. The formalization of expected utility was originally proposed by Danuel Bernoulli [13]. This theory was popularized, in slightly different form, by von Neumann and Morgenstern [112], it is their approach I present here.

Lotteries

Let x be an outcome and X be the set containing all such outcomes. A *simple* lottery L defines a probability distribution over X such that $L(x_i) = p_i$ where p_i is the probability of outcome x occurring given the choice of lottery L. If some or all of the possible outcomes of a lottery L' are themselves lotteries, rather than concrete



(a) A simple lottery with three outcomes

(b) A compound lottery with two outcomes, one of which is another lottery

Figure 4.1. Example lotteries, the numbers near each arrow indicate the probability of the outcome

outcomes, the lottery L' is said to be a *compound* lottery. Figure 4.1 shows both a simple lottery (L1) and a compound lottery (L2).

Preference functions

When a agent compares two different lotteries (L_1, L_2), it establishes a preference relation between the lotteries. von Neumann and Morgenstern identify three different *preference* relations:

$L_1 \succ L_2$: The agent prefers lottery L_1 to lottery L_2

$L_1 \sim L_2$: The agent has no preference between L_1 and L_2

$L_1 \succsim L_2$: The agent either prefers L_1 or has no preference

There are many different axiomizations of utility theory. The following is derived from Jensen [47, 95]. Given these relations, let $M(L_x, L_y, p)$ be a mixing function which outputs a new compound lottery L' that chooses outcome L_x with probability P and outcome L_y with probability (1-p) and let $\mathbf{L}$ be the set of all lotteries, there are four axioms which are assumed to hold:

A.1 *Completeness*:

$\forall L_i, L_j \in \mathbf{L}$, $L_i \succsim L_j \vee L_j \succsim L_i$.

A.2 *Transitivity*:

$L_i \succsim L_j \wedge L_j \succsim L_k \rightarrow L_i \succsim L_j$

A.3 *Archimedian Axiom*:

$L_i \succ L_j \succ L_k \rightarrow \exists p_1, p_2 \in [0,1]$ s.t. $M(L_i, L_k, p_1) \succ L_j \succ M(L_i, L_k, p_2)$

A.4 *Independence Axiom*:

$\forall L_i, L_j, L_k \in \mathbf{L}$ and $p \in [0,1], L_i \succsim L_j \leftrightarrow M(L_i, L_k, p) \succsim M(L_j, L_k, p)$

These axioms form the basis for establishing a preference ordering over lotteries. The first axiom simple states that the relation covers all possible pairs of lotteries. For any pair of lotteries, L_i, L_j, L_i is preferred to L_j, or L_j is preferred to L_i or the agent has no preference. Together with the second axiom it establishes a *weak ordering* over lotteries. The third axiom essentially states that no ordering is fixed based purely on outcomes, there is always a manipulation of probabilities which can alter the preference ordering of the lotteries. The final axiom establishes independence. It states that if the agent has a preference between two lotteries, then either lottery can be mixed with a third lottery without affecting this preference. This also implies the ability to substitute equivalent lotteries without affecting the preference ordering.

The Utility Function

The utility function U, provides a numeric *utility* value for lotteries which correspond to the agent's preference ordering. If the agent prefers or is indifferent to L_i versus L_j, then the utility of L_i is greater than or equal to the utility of L_j. The definition of the utility theorem is established in the following theorem:

Theorem 1 (*von Neumann and Morgenstern 1947*)**.**
An agents preference ordering $\succsim$ *satisfies axioms A.1-A.4 if and only if there exists a* **utility** *function U such that:*

$$L_i \succsim L_j \leftrightarrow U(L_i) \geq U(L_j) \tag{4.1}$$

$$\forall p \in [0,1], U(M(o_i, o_j, p)) = (p)U(o_i) + (1-p)U(o_j). \tag{4.2}$$

This theorem defines the existence of the utility function for any valid preference ordering of lotteries.

Expected Utility

The previous definition of the utility function only handles the utility of lotteries. To concretely assess the utility of a lottery, it is necessary to express the utility of a lottery as a function of its constituent outcomes. This leads to the expected utility expression. Let $X(L_i)$ be the set of possible outcomes for some lottery L_i and u(x) be the elementary utility function on actual outcomes, the expected utility defines the utility of some lottery L_i as:

$$U(L_i) = \sum_{x \in X(L_i)} P(x)u(x).$$

The expression for expected utility can be seen as a corollary of the earlier von Neumann-Morgenstern theorem. For every $x_i \in X$, define a simple lottery L_{x_i} where the probability of outcome x_i is one, and the probability of all other outcomes is zero. Clearly this is a valid (if silly) lottery, by the earlier theorem, $U(L_{x_i})$ must provide a valid value. Set the elementary utility function $u(x_i) = U(L_{x_i})$. Any more complex lottery can be expressed as a compound lottery made from these simplistic lotteries L_{x_i}. For example

$$L' = M(L_{x_i}, L_{x_j}, p).$$

From the second equation in the von Neumann-Morgenstern theorem we know that:

$$U(L') = (p)U(L_{x_i}) + (1-p)U(L_{x_j}).$$

Substitution of $u(x_i)$ for $U(L_{x_i})$ produces the expected utility expression

Making decisions

The development of utility theory assumes the presence of a preference ordering and defines the utility function as implied by this preference. However, in the case of programming robots and other autonomous agents, the practical application of utility theory is actually the reverse. We develop a utility function for outcomes, and use this utility function to derive a preference ordering for the different actions available to the agent.

To use utility to compute a policy for decision making, at each possible point, the agent chooses the lottery $L = argmax_{L_i \in \mathbf{L}} U(L_i)$ Expected utility provides a means for practically applying this decision making policy. The policy results in maximizing expected utility over the lifetime of the agent. This is often called a *rational* decision making.

4.2 Utility-Guided Planning

The following describes the application of rational or utility-guided decision making to the process of motion planning. Motion planning is search of the configuration space for a successful path. Utility-guided planning is a general framework for structuring this search toward expansions with maximal expected utility.

Both because of the computational complexity of motion planning and the error inherent in sensory perception of the environment, all practical, real-world motion planners necessarily must operate with incomplete and/or inaccurate information about the state of the configuration space. However this does not mean that a planner

must operate blindly. Instead it must maximize the knowledge contained in its current information about the configuration space to guide its subsequent exploration to those areas that are most beneficial to the process of planning.

To select explorations that are most useful to the planner requires a means of evaluating the *utility* of each exploration. This utility function varies depending on particular details of the planner and the conditions under which it operates, but every utility function provides a preference ordering over possible explorations that the planner might undertake.

Before it occurs, the result of any particular exploration is necessarily stochastic. It is represented by a probability distribution over possible outcomes of the expansion, such as immediately discovering it to be obstructed, or expanding some distance. In terms of utility theory, it is a lottery L_E. Let the set E contain a set of possible outcomes resulting from some exploration by a planner, the expected utility of that expansion is given by:

$$\text{Utility}(L_E) = \sum_{e_i \in E} P(e_i)\text{Utility}(e_i)$$

In addition to the utility function, the expression for expected utility includes a function that provides the probability of a particular outcome of the exploration whose expected utility is being computed. Traditionally, this probability is a known component of the lottery. However, because the planner's knowledge of the configuration space is incomplete and inaccurate, the probability distribution over possible outcomes must be estimated from the limited subset of information that the planner has already acquired. Likewise, the utility function is approximated using the current state of the planner and its current understanding of the configuration space. Consequently, the expected utility calculation is also an approximation.

Given the means for calculating the expected utility of all possible explorations in the set X of possible explorations, a utility-guided planner applies the rational policy

which maximizes expected utility:

$$\text{PlannerExploration} = argmax_{E_i \in X} \text{ExpectedUtility}(E_i)$$

Once selected this exploration is enacted by the planner and the information obtained from the exploration is incorporated into the planner's understanding of the environment. This information progresses the planner toward the computation of a successful plan. The information also improves the planner's approximation of the probability distributions over outcomes of explorations and improves the planner's approximation of the utility function. This iterative process of guided exploration and refinement continues until a stopping criterion, such as the computation of a successful path is reached.

This incremental bootstrapping of the planning process differentiates utility-guided planning from all previous sampling-based motion planners. This differentiation provides several important advantages to a utility-guided planner. First, at each step the planner selects explorations that maximize the expected progress toward a successful plan. Second, the planner adapts to new information as it is obtained further improving its guidance of subsequent explorations.

4.3 Concrete applications to motion planning

The remaining chapters of this thesis describe the application of this general utility-guided framework to three distinct problems in robotic motion planning.

First, the utility-guided framework is applied to the problem of selecting examples in sampling-based multi-query roadmap planning. In this context, each sample is seen as a lottery with two possible outcomes; the sampled configuration is free, or obstructed. The utility function which establishes a preference ordering over samples is roadmap information gain [19]. This function evaluates the amount of information

contributed by a sample to the understanding of the complete connectivity of configuration space. The algorithm always selects the sample, or lottery, with maximal expected utility. This maximizes progress towards a complete roadmap.

Second, the utility-guided framework is applied to single-query planning. Specifically to the problem of selecting explorations used to grow branches of a random tree in configuration space. In this context, each possible direction is the lottery and the outcomes are the resulting branch which is added to the random tree. Because these approaches are single-query planners, utility is not defined in terms of global information gain, but rather in progress toward a successful motion between two configurations. By selecting the exploration with maximal expected utility, the planner maximizes progress toward a successful plan.

Finally, the utility-guided framework is applied to planning and exploring in uncertain environments. The utility-guided framework is actually applied to two separate sub-problems: the task of selecting plans and the task of selecting further sensory exploration when necessary.

When a robot enacts a plan in an uncertain environments, it is possible that an error in the plan will cause the plan to fail. The application of the utility-guided framework ensures that the plans which are selected have maximal utility, that is they minimize the expected cost of any such failure. Further, they provide a means for deciding when the expected cost of failure is too high to warrant physically executing the plan. In this context, each lottery is a possible physical action on the part of the robot, the outcomes are successful or unsuccessful physical motion on the part of the robot. Utility provides the cost (negative utility) of any failure as well as the positive utility of a successful motion.

When a path fails, the robot must use additional sensing of its environment to refine its perceptions and reduce uncertain to enable the computation of a successful path. Sensing the environment is costly for the robot in terms of energy and time.

The application of the utility-guided framework to exploration selects sensing with maximal utility to the computation of a successful plan. In this context, each lottery is an exploration, and each outcome is some information gained about the world. Maximizing expected utility minimizes the sensing required to develop a successful plan.

4.4 Heuristic Utility-Guided Planning

Motion planning is computationally intractable for many real-world robots and environments. Even sampling-based planning using uniform sampling is subject to these bounds. Even a utility-guided planner is subject to the curse of configuration space dimensionality. For example, consider utility-guided sampling for multi-query roadmap planning. A naive approach to implementing this algorithm would iterate over the entire configuration space, evaluating the expected utility of each sample. Once the entire configuration space was examined, the sample with the highest expected utility would be selected. It is clear that such an algorithm is highly impractical, even if the evaluation of expected utility were extremely fast (say on the order of 1 picosecond), evaluating the expected utility for a robot with more than a few degrees of freedom is impractical. Consequently, it is necessary to develop heuristic approaches which approximate the behavior of a planner guided by expected utility but are significantly more efficient.[1]

In all of the utility-guided planners which follow in this thesis, a principled utility-guided approach is presented. This approach is then used to design a practical heuristic method that models it.

---

[1]It is important to note that this reliance on heuristics is not unique to utility-guided motion planning. Even within the subfield of planning and search there are numerous heuristic methods for efficiently achieving solutions. For example the canonical A* [89] family of methods.

The practical heuristic utility-guided planners described in the following chapters achieve improved performance relative to existing state of the art planners. However, if this performance improvement is achieved by closely modeling the performance of a true utility-guided. This question is at least partially answered in the case of utility-guided sampling for multi-query planning, where experiments 5.6.5 in a simple two dimensional world (required to make true utility-guided sampling computationally feasible) demonstrate that the heuristic planner achieves similar, although not identical performance. While I have yet to examine a similar comparison for single-query planning or planning with uncertainty, these results suggest the possibility that these heuristic planners are similarly close to true utility-guided planning.

Chapter 5
Multi-Query Planning

5.1 Introduction

Motion planning algorithms find collision-free paths for robots in obstructed configuration spaces. Because the size of configuration space is often quite large, and in many cases the task of planning constrained by time, the efficiency of this search is a significant concern. especially in real-world motion planning. To be useful for real world robotics, a motion planner must be able to compute a path quickly. The actual time required depends on the robots task and how quickly the environment changes, but is general on the order of fewer than ten seconds. The following describes utility-guided sampling, an application of the utility-guided framework to sampling-based roadmap motion planning. Specifically to the task of selecting the samples used by the motion planner to construct its approximation of configuration space connectivity.

To plan efficiently despite the proven computational intractability of motion planning, sampling-based methods compute an approximate implicit representation of configuration space connectivity. This representation is constructed by sampling and observing a subset of all points in a particular configuration space. These methods are discussed in detail in Section 2.2.2

The efficiency of a multi-query sampling-based algorithm is determined by the time it requires to construct a sufficient approximation of configuration space connectivity. Intuitively, this is a representation capable of computing a path between any pair of configurations for which a feasible path exists. Section 5.2 provides a formalization of this notion. While planning, a sampling-based algorithm constructs

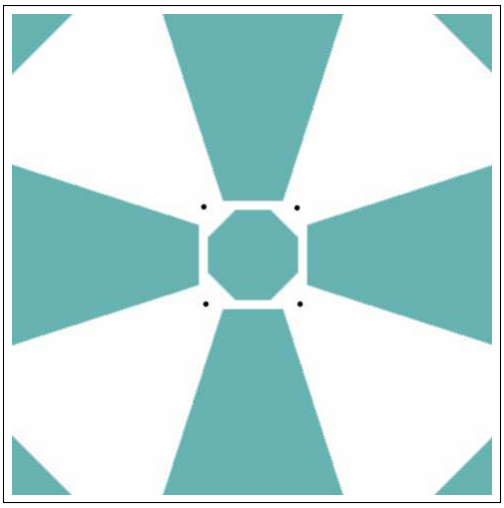


Figure 5.1. An optimal set of configurations that provide a sufficient representation of a simple configuration space. A roadmap constructed from the four configurations shown can compute any feasible path in the configuration space.

an approximation of the sufficient approximation. The planner's approximation of the configuration space is a direct function of the configuration space samples that the planner observes. Consequently, sampling is the search for the set of samples that provide enough information to construct a sufficient approximation of configuration space connectivity.

For every configuration space, there is an optimal number of samples that must be selected to construct a sufficient approximation of configuration space connectivity. As an example, consider the simple, two-dimensional configuration space in Figure 5.1. Selecting each of the four configurations shown results in a complete roadmap of the configuration space. A sampling-based planner with optimal performance simply selects each required sample. Unfortunately, identifying the set of optimal samples requires complete knowledge of the boundaries of the configuration space obstacles. Information, which if known, would imply knowledge of the configuration space's connectivity. This means that all of the information necessary for selecting the optimal set of samples is unavailable to the planner. Though the planner begins with no information about the configuration space, each subsequent exploration of the con-

figuration space provides information to the planner. This information improves the configuration space roadmap. This information can *also* guide subsequent exploration of the configuration space. This utility-guided exploration iterates exploration and incorporation of the resulting new information until a sufficient representation of the configuration space is found.

The utility-guided framework is also motivated by the theoretical analysis of multi-query sampling-based planning. This analysis shows that the expected number of samples required for a complete representation of configuration space connectivity can be expressed in terms of the probability of sampling free configuration space and the ability to select samples that are useful to the motion planner. By reducing the value of each of these factors, utility-guided sampling improves the expected runtime of the planner. These theoretic improvements are corroborated by experimental results in a variety of planning problems.

5.2 Analyzing multi-query planning performance

This section describes a theoretical analysis of sampling-based motion planning that provides a framework for understanding and improving sampling-based planner performance. Sampling-based planners sample a series of configurations until a complete approximation is constructed. Consequently, the analysis considers the expected number of samples required to construct such an approximation. The expected number of samples is a proxy for the expected runtime of the motion planner, the value of interest. It is important to note that this assumes that on average, the computation associated with observing each configuration and the edge checking required to connect the configuration to the roadmap is constant. In practice this may not actually be true. It remains an open question to determine how to develop an improved analysis of planner runtime. This analysis is closely related to a similar analysis of Ladd et al. [58] that considered single-query sampling-based planning.

5.2.1 Notation

The following describes the notation used to develop a theoretical analysis of the performance of sampling-based motion planning. Recall that the configuration space is the union of the sets of free and obstructed configurations of a robot:

$$C = C_{\text{free}} \cup C_{\text{obstructed}}.$$

Every configuration space defines a relation R_C that represents the connectivity of the configuration space to some precision ϵ. R_C is a recursive relation defined as follows:

Base Case:

$$(q_x, q_y) \in R_C \iff q_x, q_y \in C_{\text{free}} \cap (\text{distance}(q_x, q_y) < \epsilon)$$

Recursive Case:

$$(q_x, q_y) \in R_C \exists q' \in C_{\text{free}} (q_x, q') \in R_C \wedge (q', q_y) \in R_C)$$

A motion planning algorithm computes a relation $\hat{R}$ that approximates R_C. Ideally, $\hat{R}$ equals R_C. To construct $\hat{R}$, the planner repeatedly samples a configuration q and uses information obtained from sampling q to update its current $\hat{R}$. An algorithm for constructing the approximation $\hat{R}$ is given in Figure 5.3.

To further understand the operation of a sampling-based motion planning algorithm, I introduce a new relation $q_x \leftrightarrow q_y$, which indicates that there is a collision-free straight-line path between q_x and q_y. I also introduce a collection of sets over configuration space that I call *connected regions*. Each member set, $p_0 \ldots p_k$, obeys the rule $q_x, q_y \in p_i \rightarrow q_x \leftrightarrow q_y$.

The set of connected regions, P is a set of sets. The connected regions form are a tiling of C_{Free}, that is:

$$\cup_{p_i \in P} p_i = C_{\text{Free}}$$

Given any particular p_i, I further subdivide p_i into a set of regions p_{ij} consisting of all points $q_i \in p_i$ s.t. $\forall q_j \in p_j, q_i \leftrightarrow q_j$. Let P' be the set of all p_{ij}. The set P' for a simple 2-d configuration space is shown in Figure 5.2.

5.2.2 Expected samples required for planning

To understand the runtime performance of a sampling-based planner we must understand the expected number of samples required to construct a sufficient approximation of configuration space. I begin by examining the exploration required to build a sufficient approximation of configuration space connectivity. Next I calculate the expected number of samples required to achieve this exploration.

The analysis begins with a simple lemma:

Lemma 1 1. *Given the multi-query sampling-based planning algorithm, sampling one configuration in each connected region p_{ij} is sufficient for constructing a sufficient approximation of R.*

Proof. To prove this, I must show that for any pair of configurations q_m, q_n, if $(q_m, q_n) \in R_C$, then $(q_m, q_n) \in \hat{R}$.

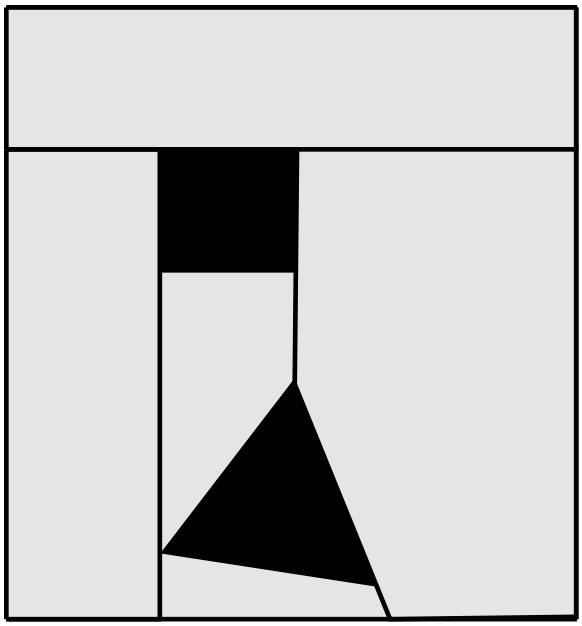
There are three cases for this proof:

distance$(q_m, q_n) < \epsilon$

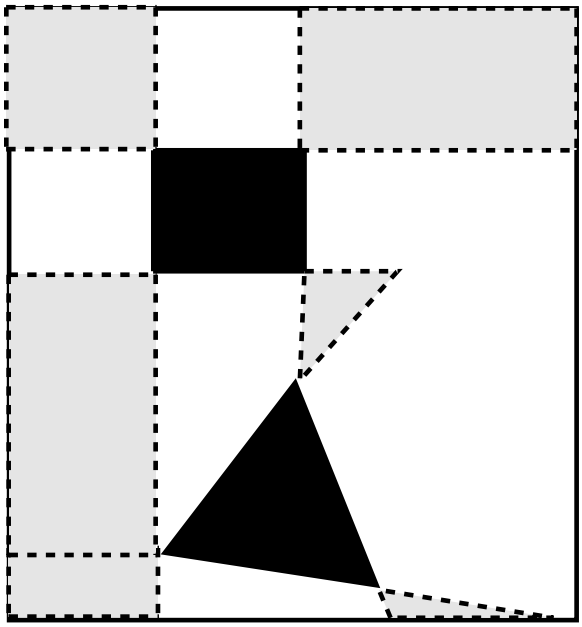
In this case, (q_m, q_n) is in $\hat{R}$ by definition.

$q_m \in p_i$ **and** $q_n \in p_i$

In this case, because I have at least one sample in p_i call this sample q_i, $(q_m \leftrightarrow q_i)$ and $(q_i \leftrightarrow q_n)$ therefore, (q_m, q_n) must be in $\hat{R}$.



(a) The set of connected regions P



(b) The set of regions P'

Figure 5.2. The set of connected regions P and P'

MultiQueryPRM

R = empty roadmap graph
foriterations
q = ChooseConfiguration
if($q \in C_{\text{free}}$)
R.addVertex(q)
foreach q' $\in$ neighbors(R, q)
if($q \leftrightarrow q'$)
R.addEdge(q', q)
return R

Figure 5.3. The basic multi-query sampling-based roadmap algorithm

$q_m \in p_i$ **and** $q_n \in p_j$

This case is more complicated and requires induction to solve. I will perform induction on the number of p_{kl} that lie between p_i and p_j. The base case is that there are zero p_{kl} between p_i and p_j, in this case then there must exist p_{ij}, by the assumption I must have drawn a sample in this region, let this sample be q_{ij}, I know that $(q_m \leftrightarrow q_{ij})$ and $(q_{ij} \leftrightarrow q_n)$ therefore, (q_m, q_n) must be in $\hat{R}$. Thus the base case is proved.

For the recursive case, assume that this is true for n or fewer intervening regions. Let q_{kj} be a sample in some region p_{kj} such that $(q_m, q_{kj}) \in R_C$, because q_m and q_n are connected, I know such a sample must exist. By induction I know that (q_m, q_{kj}) must also be in $\hat{R}$ and since (q_{kj}, q_n) is also in $\hat{R}$, then (q_m, q_n) must be in $\hat{R}$ as well, thus proving the inductive case.

□

Given this lemma I now have a definition of the problem, namely: What is the expected number of samples required to sample at least once in every p_{ij}? There are clear connections between this problem and the *coupon collector's problem* that have been explored elsewhere in the context of single-query planning [58]. In the analysis of

sampling-based motion planning, each "coupon" corresponds to a connected region of configuration space. These regions necessarily have different sizes and thus different probabilities of being sampled. It is clear that this more general expression provides a more accurate estimate of the expected samples required to develop a complete approximation of configuration space connectivity. Given some set of coupons C that an actor samples with replacement, the coupon collector's problem calculates the expected number of coupons that must be sampled to obtain a complete set. Note that because coupons are sampled with replacement, this expected value is greater than $|C|$.

If the probability of sampling each coupon is identical, then the expected number of coupons is a classical exercise whose solution is $n \log n$ when there are n coupons. This is the assumption used by Ladd et al. [58] in their analysis of single query planning. However, this assumption limits the analysis to being an upper bound rather than an exact calculation of the expected number of samples. A more complete analysis of the coupon collectors problem [32] provides the expected number of samples in the general case when each coupon has different probability:

$$\sum_{x=0}^{n-1}(-1)^{(n-1-x)}\sum_{y=0}^{x}\frac{1}{P(\text{Coupon}_y)}.$$

The coupon collectors problem provides the expected number of samples required to obtain all n coupons. However, this assumes that each sample drawn from the configuration space corresponds to sampling a coupon/connected region. Clearly, this is not the case. Samples in the set $C_{\text{obstructed}}$ are not in connected regions and thus do not provide coupons. To obtain the actual expected number of coupons, a planner must examine more samples. Through an application of the binomial theorem [58] and the probability that a sample drawn by the planner is inside C_{free} ($P(q = \text{free})$), the true number of expected samples can be calculated.

$$\begin{aligned}
\text{ExpectedSamples} &= \frac{1}{P(q = \text{free})}\text{ExpectedCoupons} && (5.1)\\
&= \frac{1}{P(q = \text{free})}\left(\sum_{x=0}^{n-1}(-1)^{(n-1-x)}\right. && (5.2)\\
&\quad \left.\sum_{y}^{x}\frac{1}{P(\text{Coupon}_x)}\right) && (5.3)\\
& && (5.4)
\end{aligned}$$

Despite this formal expression, in practice it is difficult to calculate the expected number of samples required to completely approximate configuration space connectivity. This is because identifying the exact number and size of the connected configuration space regions requires understanding the boundaries of all configuration space obstacles. Consequently, the previous theoretical analysis is significantly more useful as a relative measure of the impact of particular sampling strategies on the runtime of multi-query motion planning. This analysis enables the development of several theorems that compare the expected runtime of different sampling strategies.

Theorem 2. *For any given configuration space, if two sampling strategies S_1 and S_2 have the same probability of sampling any particular connected region given that the sample is in C_{free} then the sampling strategy with the higher probability of sampling free configuration space produces a more efficient motion planning algorithm.*

Proof. If both sampling strategies have the same probability of sampling any particular connected region given a sample in free configuration space, then both sampling strategies require the same expected number of coupons. Without lack of generalization, I can assume that S_1 is more likely to sample free configuration space. Because both strategies require the same expected number of coupons, I can conclude that the expected number of samples for S_1 is less than the expected number of samples for

S_2, since $1/P(q \in C_{\text{free}}|S_1) < 1/P(q \in C_{\text{free}}|S_2)$. By assuming the expected number of samples is proportional to the expected runtime of the motion planning algorithm I have proved that sampling using S_1 produces a more efficient planner. □

Theorem 3. *Define a "history-less" sampler to be a sampling strategy that does not use knowledge of its current approximation of configuration space connectivity ($\hat{R}$) when sampling. For some such sampling strategy S in a configuration space C with n connected regions, the minimum expected number of samples required by such a strategy is:*

$$\frac{1}{P(q \in C_{free}|S)} n \log n$$

Proof. Because a history-less sampler does not have knowledge of the current set of coupons it possesses, it must select coupons at random. Given such a strategy, the minimal expected number of coupons required by the sampling strategy is when every coupon has equal probability. Thus S is expected to require $n \log n$ coupons. When this expectation is combined with the inverse probability of sampling free configuration space, the theorem is proved. □

Theorem 4. *Define a "state-aware" sampler to be a sampling strategy that uses knowledge of its past exploration to direct its current sampling. For some such sampling strategy S in a configuration space C with n connected regions, the minimum expected number of samples required by such a strategy is:*

$$\frac{1}{P(q \in C_{free}|S)} n$$

Proof. A state-aware sampler knows the coupons that it has already sampled. In the optimal case, it never samples these coupons again. Thus the minimal expected number of samples such a strategy requires is equal to the size of the set of coupons or n. Combining this minimal value with the inverse probability of sampling free configuration space, the theorem is proved. □

Corollary 1. *If an optimal state-aware sampler and an optimal history-less sampler have the same probability of sampling free configuration space, the optimal state-aware sampler has a faster expected runtime.*

Proof. This follows directly from Theorems 2 and 3. Because the probability of sampling free configuration space is the same, the optimal state-aware sampler has an expected runtime which is less than the expected runtime of history-less sampler. □

The previous results are of significance to real-world motion planning because they provide guidance for the development of efficient motion planning strategies. The first theorem shows that sampling strategies should attempt to minimize the probability of selecting a sample from obstructed configuration space. The second and third theorems show that planners that are aware of their past exploration and use this knowledge to guide subsequent search can achieve improved performance by a factor of $\log n$ where n is the number of connected regions in the configuration space. Since for many real-world configuration spaces, n can be quite large, this is a significant potential for improvement. These theoretical results provide a framework and inspiration for the development of utility-guided sampling.

Utility-guided motion planning is inspired by the theoretical analysis previously introduced in Section 5.2. This analysis shows that a successful exploration of configuration space selects a subset of configuration space containing a sample from each member of the set of connected configuration space regions. A generalized form of the coupon collectors problem shows that the expected number of samples required to obtain this complete set is expressed as a product of the number of connected regions and the inverse of the probability of sampling free configuration space. Sampling configurations and incorporating them into the roadmap is the significant computational cost of sampling-based motion planning. Reducing the expected number of samples required by the motion planner will result in a more efficient planner. To reduce the expected number of samples required to develop a complete approximation

of configuration space, I examine the two factors in the expected samples expression (Equation 5.4) individually.

Consider the probability of sampling free configuration space. For a uniformly random sampler, the probability of sampling free configuration space is $|C_{\text{free}}|/|C|$. If the sampler has perfect knowledge of obstructed and free space, this probability is simply $|C_{\text{free}}|/|C_{\text{free}}| = 1$. In general, reducing the probability of selecting obstructed samples improves the performance of the resulting exploration strategy.

The other factor in the expected number of samples is the expected number of connected regions that must be sampled by the planner. It is important to note that this is *not* just the number of unique connected regions, but includes re-sampling a particular connected region multiple times. Most often this occurs because the sampler ignores its history of previous explorations. For example, the bridge sampling strategy [38] is equally interested in "bridges" in configuration space whether or not they have previously been explored. The same is true of all locally heuristic sampling strategies. By maintaining a history of past exploration and using this knowledge to direct future exploration, the expected number of coupons required is reduced from $n \log n$ to simply n. Unfortunately, this optimal reduction requires perfect knowledge of the configuration space. However, even partial knowledge obtained incrementally as the planner explores provides a reduction in the expected number of coupons required by the planner.

Intuitively, reducing the first factor corresponds to predicting the outcome of future sampling. The planner uses past information to reduce the probability of sampling obstructed configurations. Reducing the second factor requires understanding the planner's current approximation of configuration space connectivity. If the planner knows what regions (or "coupons") it has already explored, then it can focus future exploration on unknown regions. I will describe that the utility-guided strategy accomplishes reductions in both of these factors.

Ultimately, the goal of a motion planning algorithm is to develop a representation of configuration space connectivity that enables the computation of all possible paths in the configuration space. I can measure a planner's progress toward this goal by examining its *coverage*. The coverage of a multi-query motion planner is the percentage of all valid paths in the configuration space the motion-planner can currently solve. Planners that operate in time-constrained environments (such as the real world) should also maximize their coverage for any arbitrary number of samples. Such an *anytime* planner, is guaranteed to maximize its performance even if its computation is halted prior to constructing a complete representation of the configuration space.

Such a planner requires a way to estimate the increase in coverage that results from sampling some arbitrary configuration q. I call the function that returns this estimate the *utility function* because of its similarity to traditional utility theory. Given the planner's current approximation of configuration space connectivity, the configuration's location and a state (obstructed or free) for the configuration, the utility function estimates the increase in planner coverage resulting from sampling that particular configuration.

The utility function is defined in terms of the state of a configuration. When deciding which configuration to sample the planner does not know the state of the configuration. A solution to this problem is to develop a predictive model of the state of the configuration space and calculate the *expected utility* function for the configuration space. The expression for expected utility is:

$$U_{\text{exp}}(q) = \sum_{x \in \{\text{free,obstructed}\}} \text{Utility}(q = x)\ P(q = x). \tag{5.5}$$

In practice, I assume that the utility of an obstructed configuration is zero. This is not entirely true since obstructed configurations improve the predictive model of

the configuration space. Unfortunately, this improvement is small and difficult to quantify. Thus the expression for expected utility simplifies to:

$$U_{\text{exp}}(q) = \text{Utility}(q = \text{free})\ P(q = \text{free}). \tag{5.6}$$

Given the equation (5.6) for expected utility, the utility-guided planner biases its sampling in proportion to the expected utility of the samples.

Relating this expression back to the previous theoretical analysis of sampling-based planning, We see that predicting the state of a configuration $P(q = x)$ predicts the future outcome of sampling. This reduces the value of the first factor in the expression for expected runtime. The utility of the configuration, $(\text{Utility}(q = x))$ reflects the current state of the planner. This reduces the second factor in the expected runtime. Consequently, utility-guided sampling provides theoretic improvements in the expected performance of multi-query planning.

The following sections describe how this theoretical improvement can be embodied in a practical implementation of utility-guided planning. The first section describes roadmap information gain, the choice of utility function. The second describes the models used to predict the state of the configuration space. The final section provides the practical algorithm for utility-guided sampling used for empirical analysis.

5.3 Utility Functions

At the heart of utility-guided sampling is the utility function. The utility function characterizes the value of each configuration. Because the planner acts to maximize this value, the utility function must closely correspond to actual progress toward the planner's goal. Utility is calculated using the planner's current state, i.e. the current roadmap and previously observed configurations. Consequently, a sampling strategy that maximizes utility adapts to reflect the current needs of the planner. This means

that not only does utility guide sampling towards regions of configuration space that are inherently more useful (e.g. narrow passages), it also guides sampling away from areas whose utility is low because their connectivity is already well understood.

The goal of the planner is to understand the connectivity of the configuration space. One way to measure the utility of a sample is to measure amount of information that sample contributes to this understanding of connectivity. Previous work on entropy-guided sampling [19] uses the principles of information theory [97, 98] to formally define the information gain for configuration space roadmaps. In this work, I use a utility function that is equal to the information gain each configuration provides about the connectivity of the configuration space.

5.3.1 Roadmap Entropy and Information Gain

Information gain represents the change in the entropy of a system as a result of gaining knowledge related to the system. Given some system S, some new knowledge K, $H(S)$, the entropy of the system prior to observing K, and $H(S|K)$, the entropy of the system after observing K. The information gain resulting from K is:

$$IG(S|K) = H(S) - H(S|K).$$

For motion planning, the system is the roadmap R and the new information is the observation of some unobstructed configuration q. I use the information gain provided by the configuration as the utility function.

Utility-guided motion planning maximizes its utility-function, in this case information gain. Maximizing information gain corresponds to maximally reducing entropy. Consequently, I must define the measure of roadmap entropy such that it has minimal entropy when the roadmap is a complete approximation of configuration space connectivity. A utility-guided planner using this definition of entropy maximizes its progress toward a complete approximation of the configuration space. Entropy is

formally defined over some probability distribution, $P(x)$. Given such a probability, the entropy of the distribution is:

$$-\sum_{x} P(x) \log P(x).$$

Thus, to define the measure of roadmap entropy I must define a probability distribution whose entropy I can measure.

At any time, a configuration space roadmap consists of a number of disconnected components. Each of these components has a visibility region containing all free configurations with an unobstructed straight-line path to a node in the component. For my purposes I restrict these regions to be a strictly disjoint set by assigning any configuration in the visibility region of multiple components to the visibility region of the nearest component. Given this set of configuration space regions, I use the probability distribution that a particular free space sample drawn uniformly at random will lie in a particular component's visibility region. This distribution has the desired characteristics for defining roadmap entropy. When the roadmap is fully connected, there is only a single component and the entropy of the distribution is zero. When there are a large number of different connected components, entropy is high.

Using the previous definition of entropy, I sum over all connected components in the roadmap and the entropy of a roadmap is defined to be the entropy over this distribution which is:

$$-\sum_{R_i \in R} P(R_i) \log P(R_i) = -\sum_{R_i \in R} \frac{A_i}{C_{\text{free}}} \log \frac{A_i}{C_{\text{free}}}$$

Adding some point p to a roadmap R will result in a new roadmap R' which will contain a new connected component R_u which consist either solely of p or a combination of p with several connected components already in R. In either event,

if I define R'' to be the (possibly empty) set of components which are joined by the point p, then I can define the new roadmap R' to be those components which are in R but not in R'' unioned with R_u. Given this I can define the information gain provided by adding some point p which results in the combination of connected components create R_u:

$$\begin{aligned} IG(R,q) &= H(R) - H(R') \\ &= -\sum_{R_i \in R} \frac{A_i}{C_{\text{free}}} \log \frac{A_i}{C_{\text{free}}} - -\sum_{R_j \in R'} \frac{A_j}{C_{\text{free}}} \log \frac{A_j}{C_{\text{free}}} \\ &= \sum_{R_i \in R} -\frac{A_i}{C_{\text{free}}} \log \frac{A_i}{C_{\text{free}}} + \sum_{R_j \in R'} \frac{A_j}{C_{\text{free}}} \log \frac{A_j}{C_{\text{free}}} \\ &= \frac{1}{C_{\text{free}}} - \sum_{R_i \in R} A_i \log \frac{A_i}{C_{\text{free}}} + \sum_{R_j \in R'} A_j \log \frac{A_j}{C_{\text{free}}} \end{aligned}$$

Since the area of free configuration space is constant, information gain for the addition of some point p to a roadmap R to produce a new roadmap R' is proportional to:

$$IG(R,q) \propto -\sum_{R_i \in R - R''} A_i \log \frac{A_i}{C_{free}} + \sum_{R_j \in R'} A_j \log \frac{A_j}{C_{free}}.$$

The utility of a particular sample is set equal to the roadmap information gain of that sample. Since an obstructed configuration cannot modify the state of the roadmap it offers no information and thus no utility. It is important to note that roadmap entropy, and therefore the utility-function, is dependent on the current state of the roadmap. Because the roadmap is constantly changing, the utility function is changing as well. Consequently, utility-guided sampling adapts to the changing knowledge and requirements of the planner.

5.4 Modeling configuration space

Rather than directly using the utility of the configuration space we must instead consider the *expected* utility of the configuration space. The outcome of each config-

uration space sample is stochastic. Prior to observing the configuration the planner has no way of knowing whether it is obstructed or free. The utility of a particular sample may be quite high, but if its probability of being free is very low then its *expected* utility is low. Consequently, this distribution must be incorporated into the utility-guided sampling to select samples which are actually useful. I estimate this distribution using a predictive model built from past experience. It is important to note that this use of past experience means that utility-guided sampling is adaptive. Information obtained from sampling the configuration space is continuously incorporated into the predictive model. Sampling is adapted to reflect knowledge as it is obtained. For the particulars of the predictive model I use classifiers developed in the machine learning community.

Configuration space can be viewed as a binary classification, $C(q) = 0, 1$, which takes some configuration q and returns whether or not that configuration is obstructed. Because of the topological properties of configuration space, if some q is obstructed, it is more likely that its neighbors are also obstructed. The same is true of free configurations. Given a collection of sampled configurations that have been labeled with their state, I can use learning methods from machine learning [75] to construct an approximation of the function C which I call $\tilde{C}$. This approximation function returns a number between zero and one, estimating the likelihood that a particular configuration is obstructed or free. This approximation function is the approximate model of configuration space.

There are numerous methods for constructing an approximate model of the configuration space function C. My previous work [20, 21] explores the use of mixture of Gaussian models and locally weighted regression [8]. In this work, I use a simpler K-nearest neighbor model. I have found that the computational expense of more complex models outweighs any increases in accuracy that they may offer.

Given a collection of configuration space samples Q, which have been labeled with their state, a query configuration q, which has not been observed, and $N(q,k)$, the function that provides the k-nearest neighbors in Q, I calculate $\tilde{C}$ for a particular configuration q as follows:

$$\tilde{C}(q) = \sum_{i}^{N(q,k)} C(q_i)/k$$

Note that although a complete definition of C is unavailable, it is defined for the configurations in the set Q that are already observed.

I take the output of $\tilde{C}$ to be the probability that a configuration is free. M is the current set of observed configurations.

$$P(q = \text{free}|M) = \tilde{C}(q)$$

$$P(q = \text{obs}|M) = 1 - \tilde{C}(q)$$

The Euclidean metric used by many nearest-neighbor learners have known problems as the dimensionality of the problem expands. To counteract this, I use a distance metric first applied to motion planning by Leven [65] that measures the maximum workspace distance between a set of reference points located along the robot. This metric achieves significantly higher model accuracy, especially as dimensionality increases.

5.5 Sampling algorithm

Given the predictive model and utility function, the expected utility of some configuration q is:

$$\begin{aligned} U_{\text{exp}}(q|M) &= \sum_{i \in \text{obs,free}} P(q = i|M) \cdot U(q = i, R) \\ &= P(q = \text{free}|M) \cdot U(q = \text{free}, R) \end{aligned}$$

UtilityGuidedSampling(M : Model, Roadmap : R) : q
    q := nil
    do k **times**
        q' = EntropyGuidedSample(R)
        if ($P(q' = \text{free}|M) > P(q = \text{free}|M)$)
            $q = q'$
    return q

EntropyGuidedSample(R : Roadmap) : q
    do
        C_1 := random component in R
        C_2 := random component in R, $C_1 \neq C_2$
    while (distance(C_1, C_2) > Threshold)
    q_1 := random node in C_1
    q_2 := random node in C_2
    q_n := midpoint(q_1, q_2)
    for each d in D
        $q_r[d]$:= $q_n[d]\pm$ UniformRandom(τ)
    return q_r

Figure 5.4. The utility-guided sampling algorithm

$$= \tilde{C}(q) \cdot IG(M|q)$$

In practice, the computation of information gain for each potential sample in the configuration space is computationally impractical. However an analysis of the expression for information gain leads to several sampling principles that heuristically select configurations that maximize information gain:

Connecting a point to an existing component is better than creating a new component. Connecting a point to an existing component R_i will increase the area of that component A_i' this in turn increases the value $A_i \log \frac{A_i}{C_{free}}$ in the summation which increases the second factor, as well as the information gain.

Connecting to a larger area is better than connecting to a smaller area. Connecting to a larger area creates an even larger area, with a correspondingly larger probability

of selection, as this probability increases the overall entropy of the system drops more than if two probabilities became closer (as they would if a point was added to a smaller connected component), so prefer adding points to larger components.

Connecting the largest volume of previously disjoint connected components will maximize the decrease in entropy of this distribution. Not only does it reduce the number of connected components over which the summation is performed ($|R'| < |R|$) but it also increases the probability that configurations chosen uniformly at random will be nearest the newly merged component. As this probability increases, the log of the probability becomes closer to zero.

These principles imply a sampling strategy that attempts to connect large disjoint components. Samples on the border region between two components are likely to provide maximal information gain. This border region is difficult to compute exactly but can be approximated using hyper bounding-boxes around each connected component [19] or by selecting configurations from the area surrounding the midpoint of the line connecting two random nodes, one selected from each component. I use this second approach for the experiments with utility-guided sampling, but I have not observed significant differences between the two approximations. Other heuristic approximations such as trees of bounding boxes or oriented hyper-ellipses or clustering algorithms are also possible and bear future exploration. Since I equate information gain and utility, heuristically maximizing information gain, maximizes expected utility as well. Pseudo-code for the utility-guided sampling algorithm is shown in Figure 5.4.

5.6 Experiments

I present a variety of experiments to validate utility-guided sampling. There were three main questions addressed by these experiments:

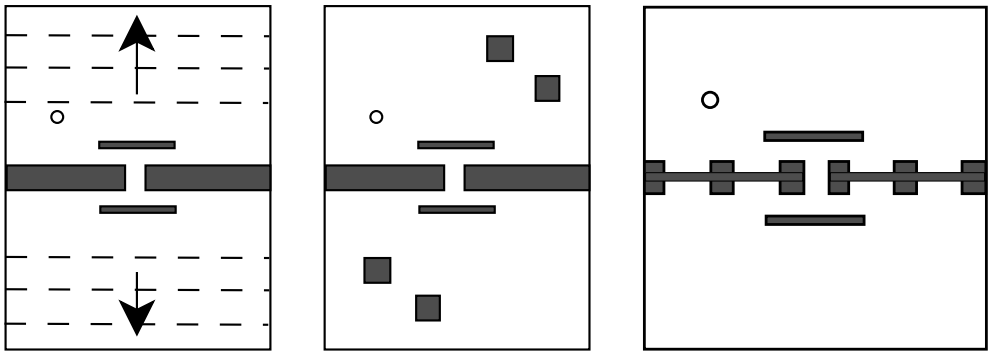
First, certain configuration space obstacles (such as narrow passages) are challenging for sampling-based planners. How does utility-guided sampling compare to other approaches when planning in configuration spaces containing such obstacles? To answer this question I performed experiments with a simple two-dimensional point robot. This enabled easy visualization and hand tuning of obstacles in the configuration space.

Second, I are interested in motion planning for real-world robots which have many more than two degrees of freedom. How does utility-guided sampling compare to other methods when planning for real robots? Does it achieve performance that is adequate for real-world planning in dynamic environments? To answer these questions, I performed planning experiments using two existing robots: the UMass mobile manipulator consisting of three-DOF base and seven-DOF arm and the UMass humanoid torso with two seven-DOF arms for a total of 14 degrees of freedom. These robots are shown in Figures 5.9 and 5.10.

Third, the interest in real-world motion planning extends to planning under time constraints. How does utility-guided sampling perform when it is forced to halt prior to computing a complete representation of configuration space connectivity? To answer this question I performed a series of experiments that measured the coverage of the planner as a function of time and compared utility-guided sampling to existing multi-query planners.

Fourth, utility-guided sampling is a synthesis of two constituent components: entropy-guided and model-based sampling. What is the contribution of each sampler? Does it depend on particular details of the robot and workspace? To answer this question we ran experiments which compared the performance of utility-guided sampling to entropy-guided and model-based sampling.

Finally, the implementation of utility-guided sampling uses heuristics to improve runtime performance. How closely do these heuristics model a sampler which ac-



(a) A challenging environment for uniform sampling. As the size of the world grows along the axis indicated the probability of sampling a relevant configuration decreases.

(b) A challenging environment for bridge sampling. Many bridges in the cspace are largely irrelevant.

(c) A challenging environment for utility-guided sampling. Numerous false openings initially appear valid in the predictive model, while the true opening seems obstructed

Figure 5.5. Challenging environments for different sampling strategies

tually calculates the expected utility of each sample? To answer this question we implemented a sampler which actually calculates expected utility and compared the performance of the two samplers.

5.6.1 Exploring Configuration Space Variation

The structure in configuration space has a significant effect on the ultimate performance of any particular planning algorithm. Various configuration space shapes such as narrow-passages [39] and bugtraps [117] have been identified as challenges for sampling-based planning. I are interested in examining how the planner performs in the presence of challenging configuration spaces. To accomplish this, I hand-designed a set of configurations spaces that presented challenges for the sampling strategies I was comparing.

Traditional PRM explores the configuration space using uniform sampling. Because of this, as the ratio of relevant configuration space to irrelevant configuration space decreases, the performance of uniform sampling degrades. To illustrate this,

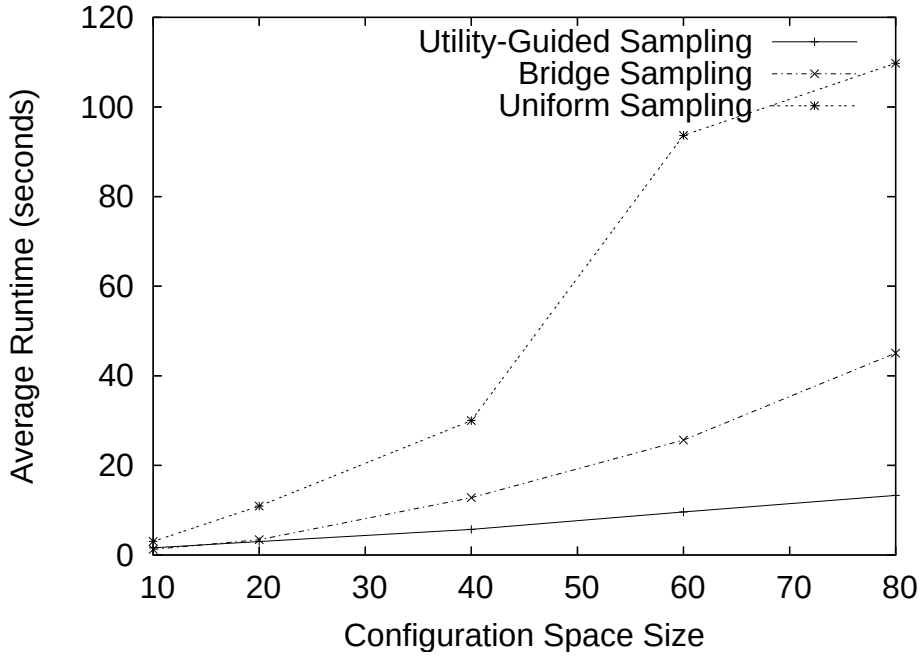


Figure 5.6. Average runtime for various planning algorithms as a function of configuration space size

I designed the two dimensional configuration space shown in Figure 5.5a. The relevant configuration space region is the "I" shaped narrow passage through the middle obstacle. As the size of the configuration space is expanded along the vertical axis (illustrated by arrows) the ratio of the area of this relevant region to the area of irrelevant configuration space decreases. From the theoretical analysis of multi-query planning I know that the expected number of samples required to solve the configuration space will increase as the size increases.

I ran traditional PRM using uniform random sampling, PRM using bridge sampling [38], and utility-guided motion planning for configuration spaces of various sizes. Each planner was run until it could motion plan through the tunnel in the wall. A graph showing the average runtime of fifty different experiments as a function of the size of the configuration space is shown in Figure 5.6.

It is clear from this graph that expanding the size of the configuration space while keeping the size of the relevant region of configuration space constant results in increasingly poor performance by uniform sampling. This is predicted by the theoretical analysis of the motion planner. As the size of the configuration space relative to the relevant region increases, the probability that a coupon is selected decreases. This quickly increases the expected number of samples required to develop a complete representation.

Heuristic sampling strategies overcome the performance limitations of uniform sampling by focusing on configurations that posseses heuristic properties that estimate the relevance of the configuration to motion planning. Numerous heuristic strategies have been proposed, an example is the bridge sampling strategy [38] which biases sampling toward configurations that lie on a straight line connecting two obstructed configurations. Although these heuristic approaches increase the probability that relevant regions are sampled, they are vulnerable to pathological behavior when irrelevant regions have the same heuristic characteristics as relevant regions. To illustrate this problem I introduced several extra bridge-shaped obstacles into the work space. These changes are illustrated in Figure 5.5b. These obstacles have little effect on the difficulty of most paths in the configuration space. In particular they do not effect the difficulty of finding a path through the wall in the middle of the world. As previously I ran 50 different experiments with traditional uniform-sampling PRM, PRM using bridge sampling and utility-guided motion-planning. The average runtime for each planner for different number of extra obstacles is shown in Figure 5.7.

From these results it can be seen that bridge sampling performs increasingly poorly as the number of irrelevant bridges is increased. At the same time, utility-guided sampling's performance stays linear. Because it focuses on global utility rather than local heuristics, it is not influenced by the irrelevant bridges. Interestingly, the performance of uniform sampling improves with the addition of irrelevant bridges. This

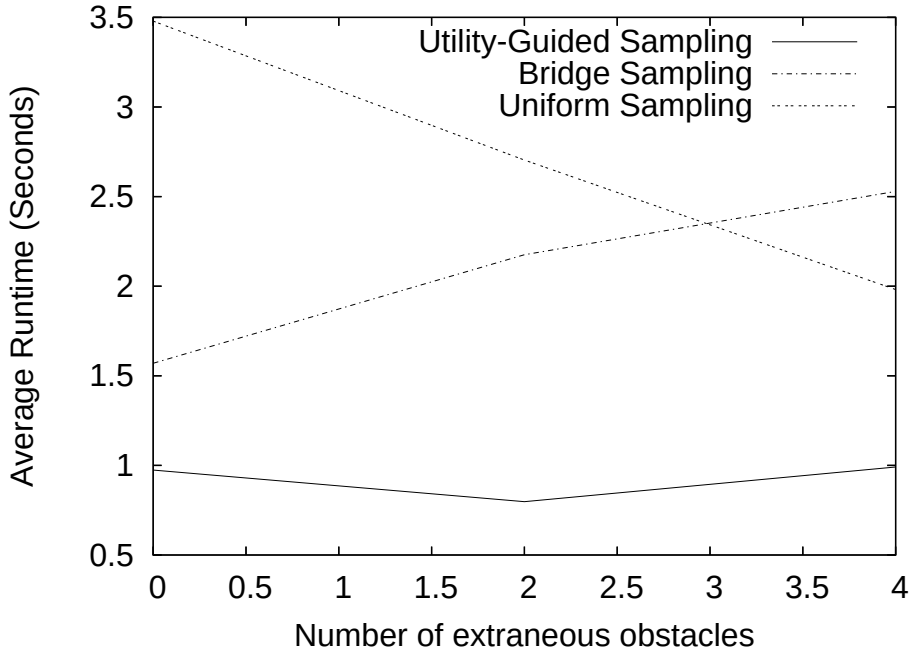


Figure 5.7. Average runtime for various planning algorithms with the introduction of extraneous obstacles

is because the introduction of irrelevant obstacles increases the ratio of relevant free space to irrelevant free space, increasing the probability that uniform sampling will select samples in the relevant tunnel and improving runtime.

I have seen that sampling strategies have pathological behavior because of their assumptions. Uniform sampling assumes that each configuration is equally relevant and its performance degrades as this assumption is increasingly violated. Likewise, bridge sampling assumes that every bridge in the configuration space is relevant. Its performance degrades as this assumption is violated. Is the same true of utility-guided sampling? The primary assumption that utility-guided sampling makes is that its model can correctly predict the structure of configuration space. Utility-guided sampling relies on this assumption for accurate assessments of the expected utility of a sample. This assumption can be violated by introducing configuration space structure that is difficult to model. The simple two-dimensional configuration space shown in Figure 5.5c is particularly challenging to utility-guided sampling. It

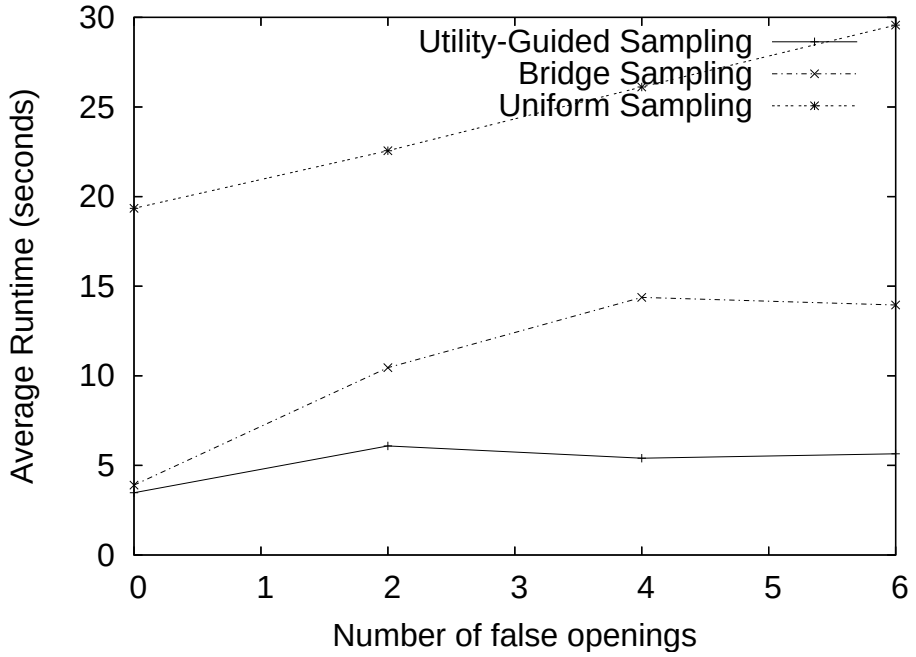


Figure 5.8. Average runtime for various planners as a function of the number of false openings

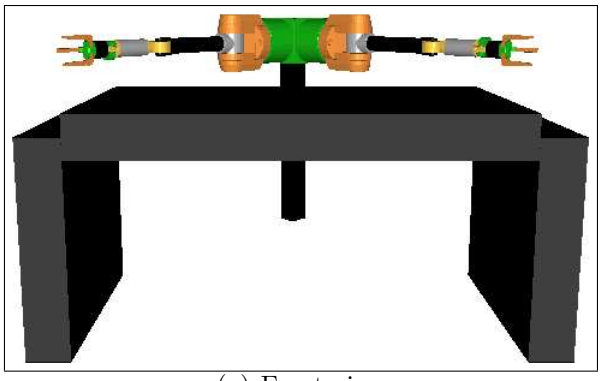
has numerous large openings in the wall that are only blocked by a narrow wall. Conversely the true passage through the wall is narrow. As a result, the initial model of this configuration space used by the motion planner will be more interested in the false openings rather than the true tunnel. However, because the utility-guided algorithm is adaptive to new information as it is obtained, its early efforts to move through obstructed configuration space will refine its model until the true passage is favored. However, this refinement slows the performance of the planner. To examine the reaction of the planners to this world I again performed experiments with uniform sampling, bridge sampling and utility-guided sampling recording the runtime required to successfully plan a path through the tunnel in the wall. A rendering of this world is shown in Figure 5.5c. The results shown in Figure 5.8 show the average runtime for fifty experiments for various numbers of false passages.

From these results it can be seen that the performance of all of the planners is influenced by the introduction of false openings. Interestingly, this occurs for separate reasons for each planner. Uniform sampling's runtime increases since the proportion of relevant free configuration space to irrelevant free configuration space decreases as false openings are added. The performance of bridge sampling is reduced because the false openings are also irrelevant bridges. Finally, the performance of utility-guided sampling also suffers because of the challenge of modeling a space with many false openings. However, because utility-guided sampling is adapts its model of the configuration space as it gains information, the impact on utility-guided performance is small. Further, the impact appears to be relatively constant with regards to the number of false openings. This shows that an adaptive strategy has a better chance at success even in environments purposely designed to challenge utility-guided sampling.

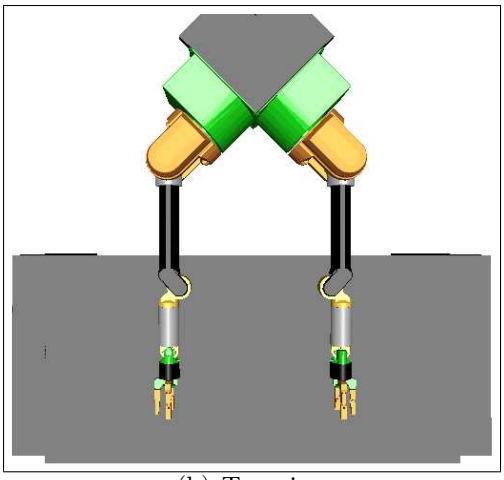
5.6.2 Real-World Motion Planning

The performance of utility-guided planning on artificial experimental worlds illustrates the benefits of this approach to planning over existing sampling strategies. However the real use of motion-planning algorithms is robots with many degrees of freedom. In particular, I am interested in applying the approaches to real-world robots capable of acting in the physical world. For these experiments I worked with the UMass Mobile Manipulator and the UMass Humanoid Torso (Figures 5.10 and 5.9). The shape of configuration spaces can not be as easily manipulated as those for a two-dimensional robot, thus the experiments I performed simply attempted to make the planning problem more or less challenging and observed how the various planners reacted to the increased challenge.

In both workspaces I made the configuration spaces more challenging by making passages in the workspace increasingly narrow. In the mobile-manipulator workspace this was achieved by adjusting the width of the hallway that the mobile manipulator



(a) Front view

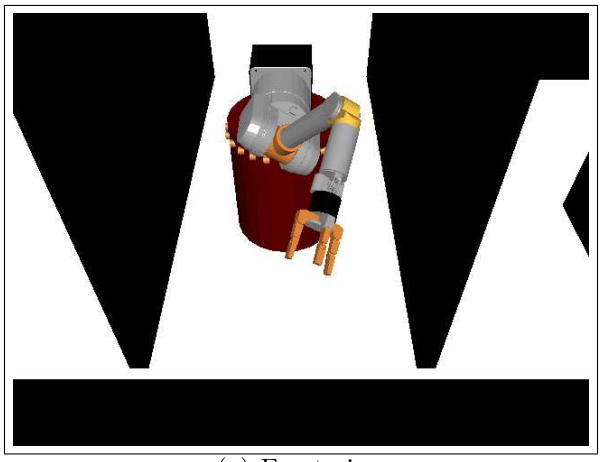


(b) Top view

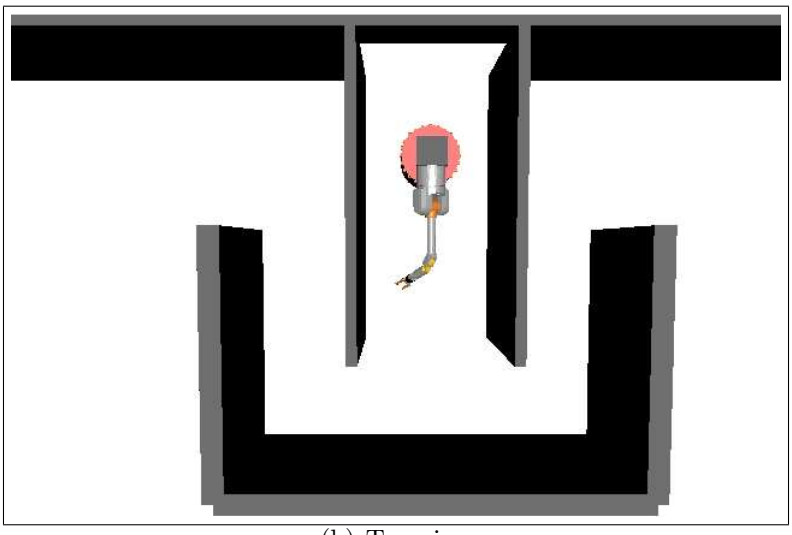
Figure 5.9. Two views of the experimental environment for the 14-DOF UMass humanoid torso

was moving through. In the humanoid torso workspace, this was achieved by moving the desk closer and closer to the torso. For all variations of the two workspaces I asked the motion planner to find a path from one side of the narrow passage to the other. For the torso, this consisted of a path from having both arms under the table to both arms above the table. For the mobile manipulator, it was a path from the doorway through the hallway to the open area below the bottom wall. The manipulator was free to choose whether it moved through the left or right hallway. For all variations and all planners I ran fifty experiments with different random seeds. The results of these experiments for both workspaces and all planners as a function of workspace difficulty is shown in Figures 5.11 and 5.12.

All of these results show that in real-world planning as the configuration space passages become narrower, utility-guided planning significantly outperforms both uniform and bridge sampling. In both domains when the passage was at its narrowest, the performance of utility-guided sampling was at least twice as fast as bridge sampling and more than ten times faster than uniform sampling. Especially in the case of the humanoid torso, these improvements in performance mean that these meth-



(a) Front view



(b) Top view

Figure 5.10. Two views of the experimental environment for the 10-DOF UMass mobile manipulator

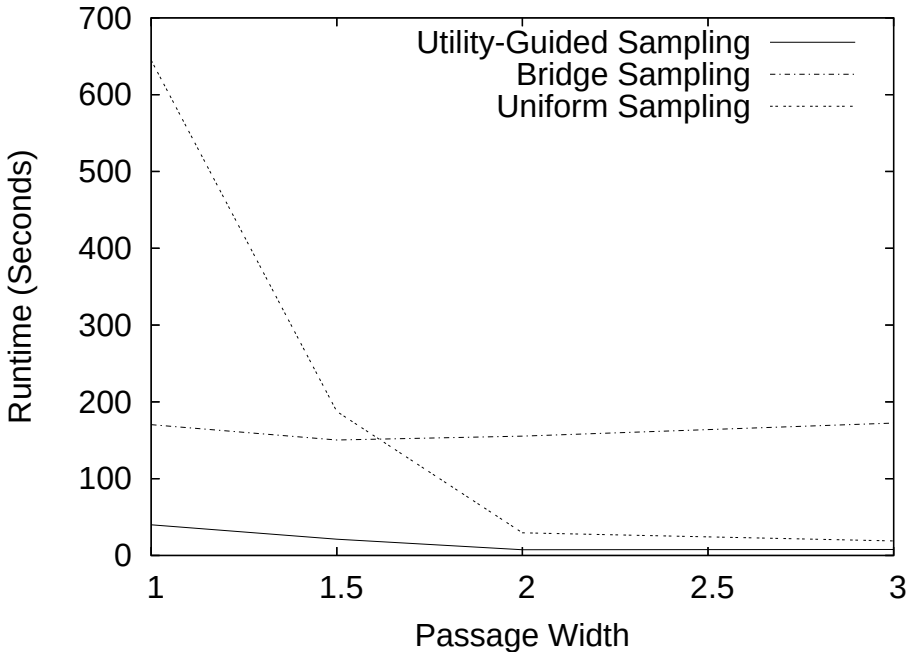


Figure 5.11. Average runtime for various planners as a function of hallway width for the UMass Mobile Manipulator (Smaller widths are harder)

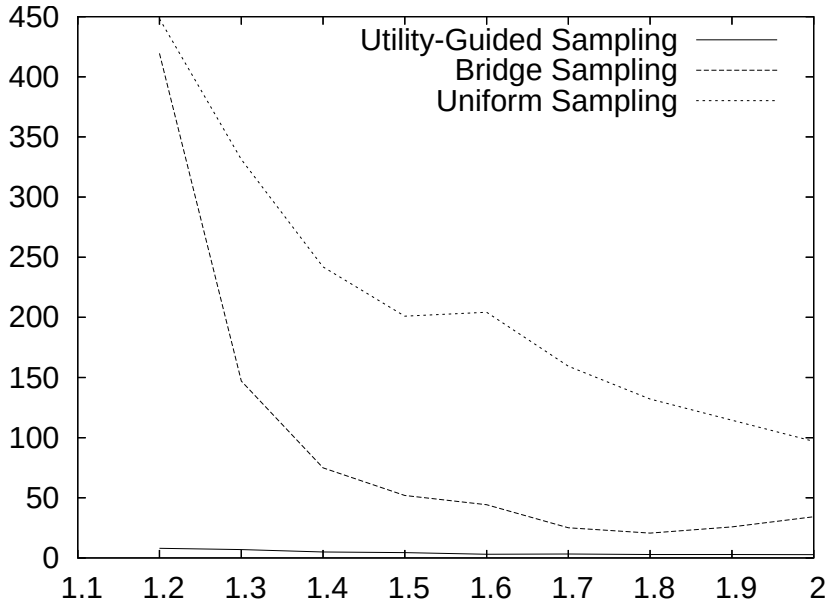


Figure 5.12. Average runtime for various planners as a function of proximity of the desk for the UMass humanoid torso (Smaller distances are harder)

ods are approaching the speeds required to perform online re-planning in dynamic environments.

5.6.3 Coverage Experiments

In addition to exploring the length of time it takes a planner to find a particular path, another important consideration for any planner is its *coverage* as a function of time. The coverage of a planner is the fraction of all feasible paths the planner can correctly compute. For any arbitrary feasible path from $q \leftrightarrow q'$, the coverage of the planner is the probability that the planner can produce a path connecting q and q'. When the planner has computed a sufficient approximation of the configuration space, its coverage is one. But how does the planner perform in the time leading up to this representation. In many real-world situations, the planner will only be able to run for a small time and then be expected to produce a plan. I would like the planner to be an *anytime* algorithm. An anytime planner maximizes configuration space coverage for any amount of time allotted for planning. For the purposes of comparison, I compare coverage as a function of time for uniform, bridge and utility-guided sampling. To assess the coverage of each planner experimentally, I ran each planner for a period of time. At regular intervals, each planner is halted and asked to compute paths between 100 pairs of configurations that were known to have feasible paths. The fraction of these feasible paths that the planner could actually compute was recorded. For each planner I ran 50 trials where the coverage of each planner was observed every half a second. For these experiments I used the UMass mobile manipulator in the world shown in Figure 5.13 which is segmented into four distinct configuration space regions. The results of these experiments are shown in Figure 5.14.

There are several trends worth noting from the results. First, utility-guided sampling and bridge sampling achieve complete coverage before uniform sampling. However, utility-guided sampling has much greater initial gains in coverage and achieves

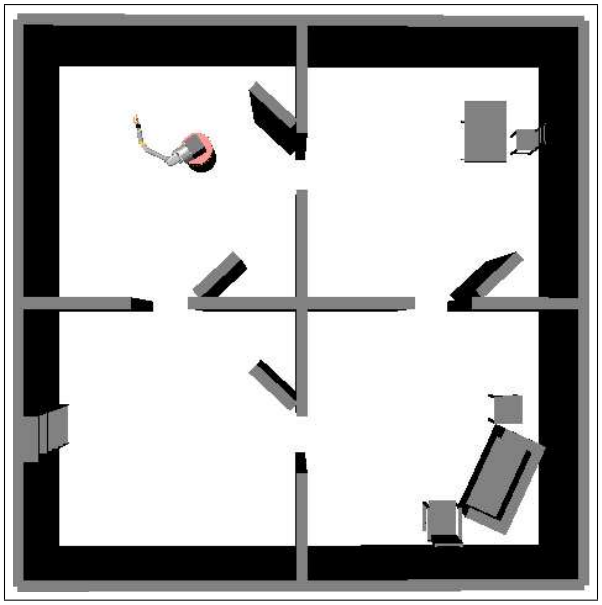


Figure 5.13. The experimental environment for coverage experiments using the UMass Mobile Manipulator

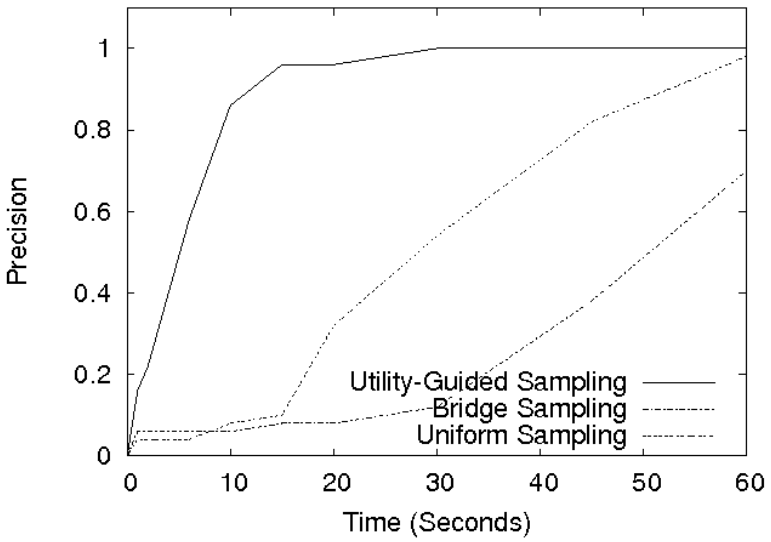


Figure 5.14. Planner coverage as a function of runtime averaged over fifty experiments

complete coverage in roughly half the time of bridge sampling. Even after only ten seconds of planning, utility-guided sampling can find compute nearly ninety percent of all feasible paths. In contrast, it takes bridge sampling nearly fifty seconds to reach the same coverage, and uniform sampling even longer. Additionally, the shape of the increases in coverage very cleanly follow a "learning curve" which indicates improved anytime performance. From this experiment it is clear that not only is utility-guided sampling more efficient at finding any particular path, but also significantly improves the fraction of paths found for a given window of planning time.

5.6.4 Comparing components of utility-guided planning

Utility-guided planning synthesizes ideas developed by two other sampling strategies: Entropy-guided sampling [19] which selects samples with maximal information gain and model-based sampling [21] which uses a predictive model and active learning to select samples. Given this synthesis, it is interesting to explore the relative contributions of these constituent sampling strategies. To do this, we ran a series of experiments comparing the performance of entropy-guided sampling and model-based sampling to utility-guided sampling. To ground the comparison, we also examined the performance of uniform sampling and the bridge [38] sampling strategy. Experiments were performed using an early virtual prototype of the UMass mobile manipulator with 4 or 6 degrees-of-freedom operating in a workspace divided in two by a long wall, a single narrow doorway is the only connection between the two halves of the workspace. This workspace is shown in Figure 5.15. Experiments were also performed using a fixed arm with 9 or 12 degrees-of-freedom operating in a workspace that is divided in two by a desk. This workspace is shown in Figure 5.16. Both of these workspaces were subdivided into two distinct configuration space regions connected by a narrow passage. Such configuration spaces are known to be challenging to sampling-based motion planners.

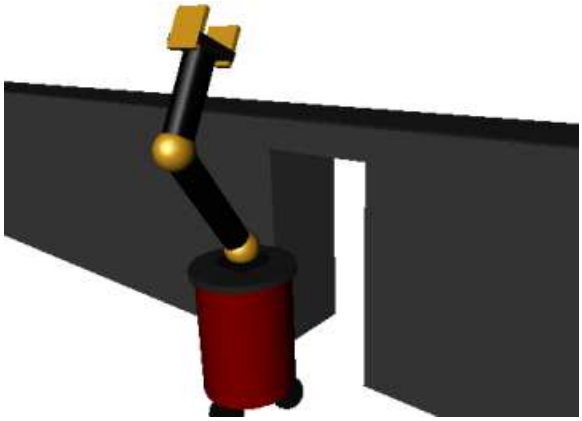


Figure 5.15. The experimental environment for the 4-DOF & 6-DOF mobile manipulator (6-DOF shown)

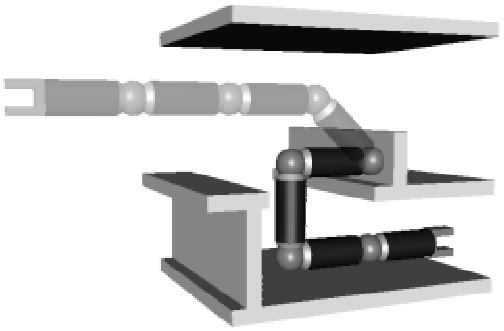
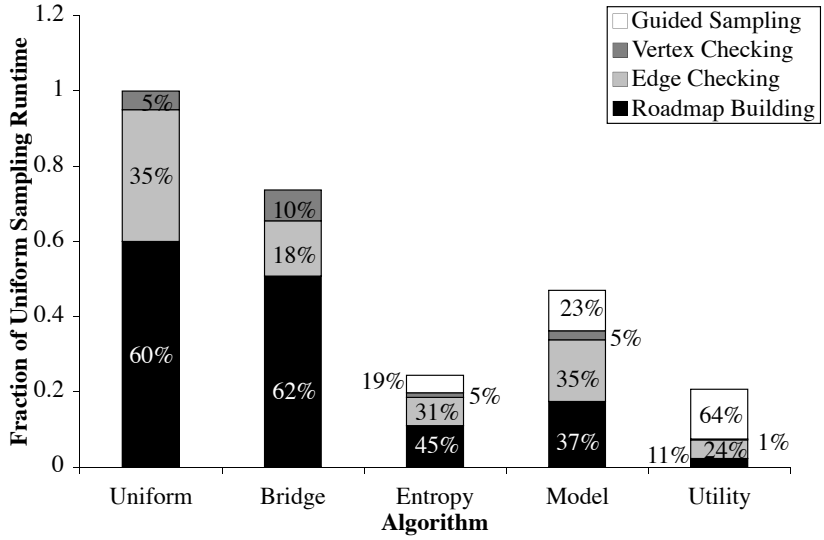
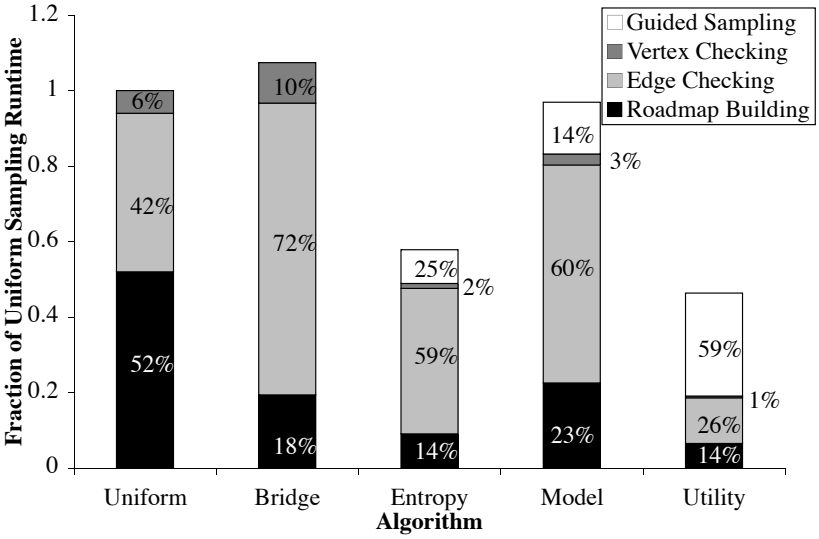


Figure 5.16. The experimental environment for the 9-DOF & 12-DOF fixed arms (12-DOF arm shown)

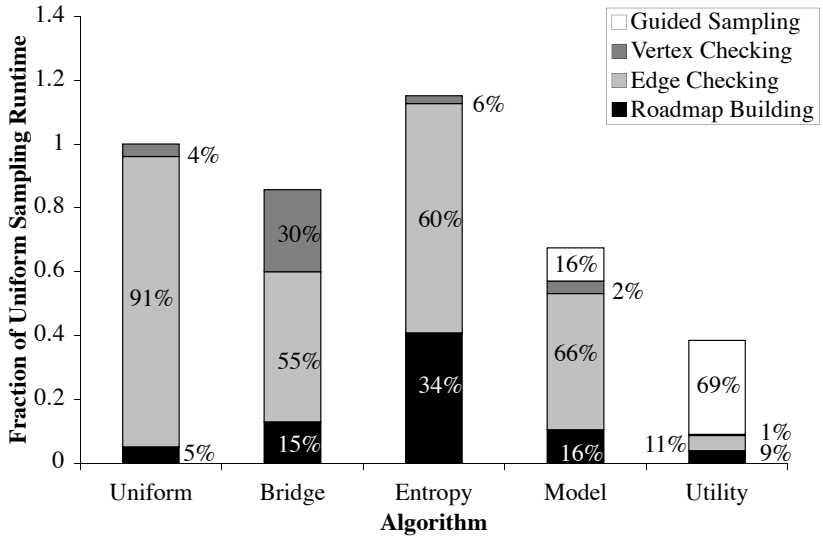
For each robot in each workspace we ran ten experiments. Each experiment consisted of a path query with start and end points randomly selected in the two different configuration space regions. Consequently, all paths required the planner to navigate the narrow passage. The average runtimes for these experiments are shown in Figure 5.17. In the graphs we also show profiling information which shows the percentage of time the algorithm spends performing particular pieces of the roadmap construction process. All runtimes are shown as a percentage of the time required for uniform random sampling to find a path through the narrow passage.



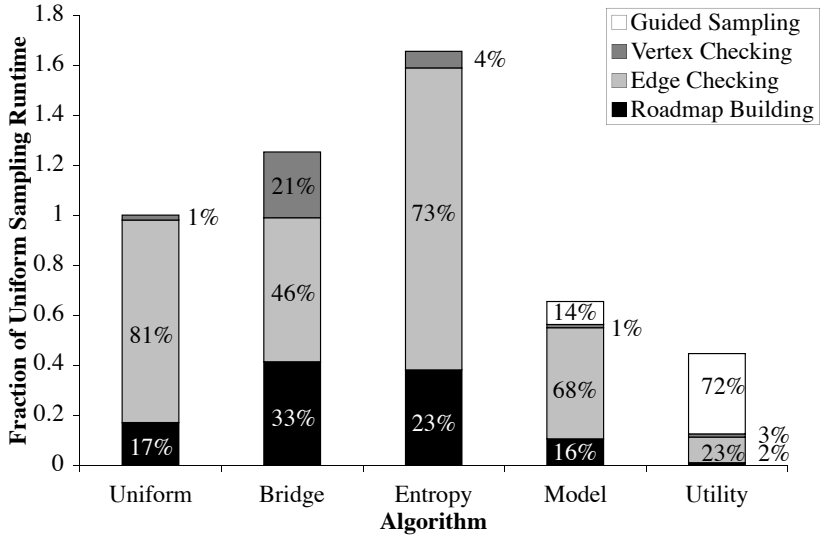
(a) 4-DOF mobile manipulator



(b) 6-DOF mobile manipulator



(c) 9-DOF arm



(d) 12-DOF arm

Figure 5.17. Runtimes for various sampling strategies as a percentage of the runtime using the uniform sampling strategy

The utility-guided sampling strategy incurs some overhead both in sampling to construct a model of configuration space and in evaluating the expected utility in order to select configurations. This overhead is included in all of the reported runtimes. In order to examine the influence of this overhead, we also profiled the operation of the planners using different samplers. In the runtime graphs, each runtime is broken down into four categories:

Collision checking

The examination of individual configurations to determine if they are obstructed or free,

Edge Checking

The examination of a series of an interpolated series of connections between two configurations to determine if a straight line path is possible,

Guided Sampling

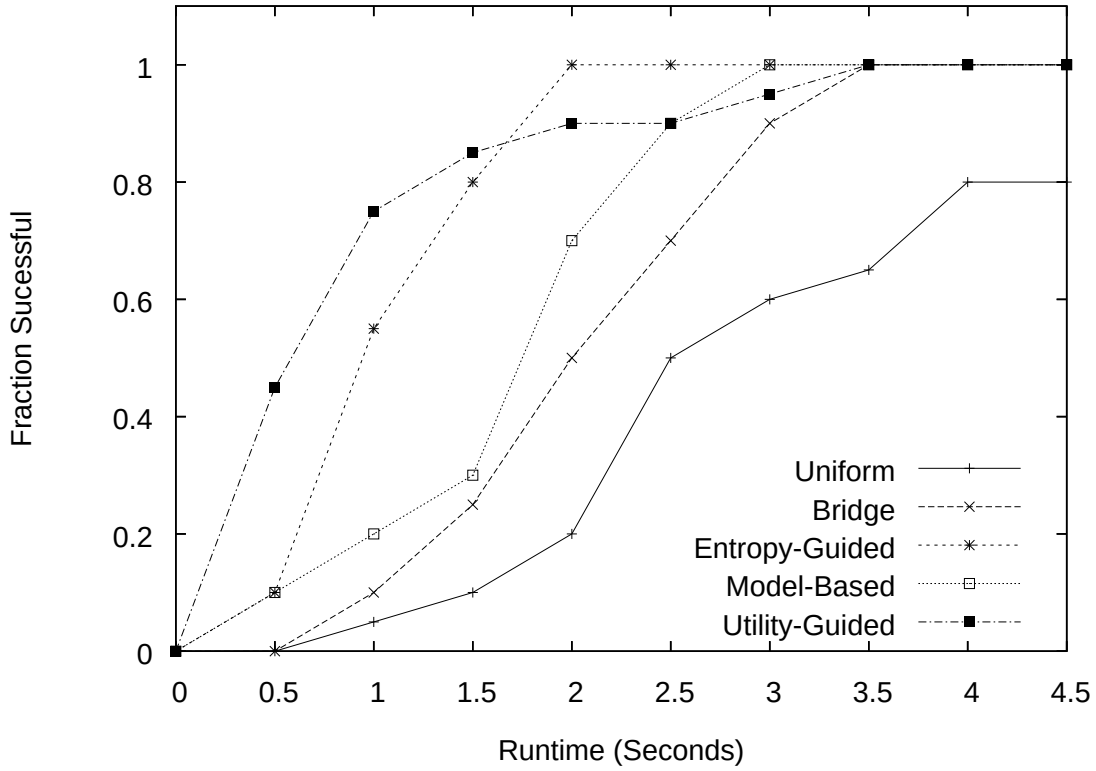
The calculation and selection of configurations guided by information from previous experience (note that this only pertains to the entropy-guided, model-based and utility-guided motion planners),

Roadmap Construction

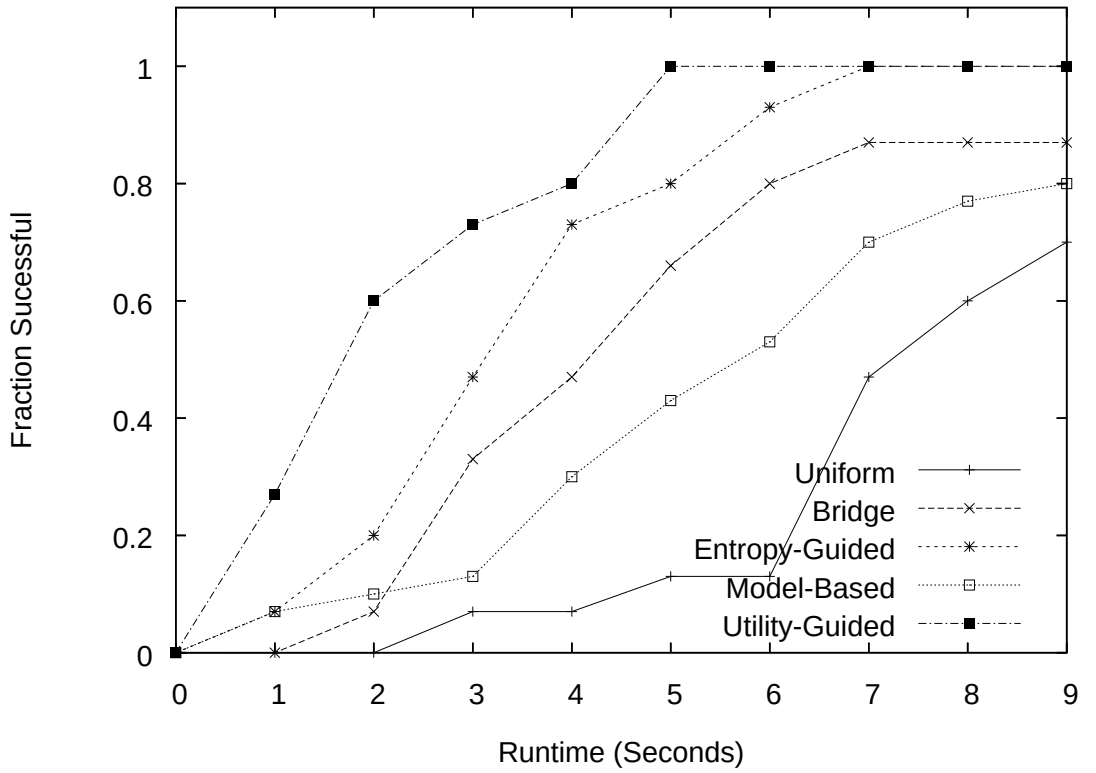
All other activities pertaining to constructing a roadmap (e.g. finding neighbors, inserting vertices/edges, etc).

It is instructive to note that although a significant portion of the runtime of the guided sampling strategies is consumed by selecting configurations. The configurations chosen are more relevant to the motion-planning process. The resulting computational savings in edge checking and roadmap construction outweighs all overhead from selecting samples.

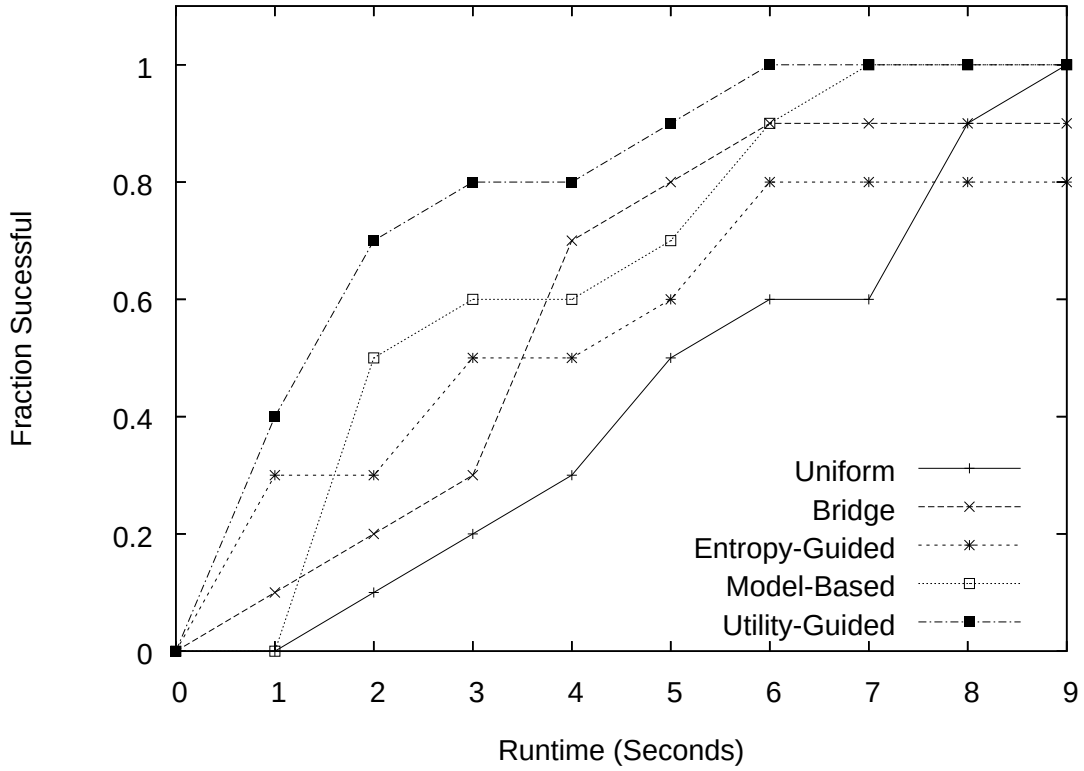
We also ran a series of coverage experiments, similar to those in Section 5.6.3, for the constituent components of utility-guided sampling. For each sampling strategy



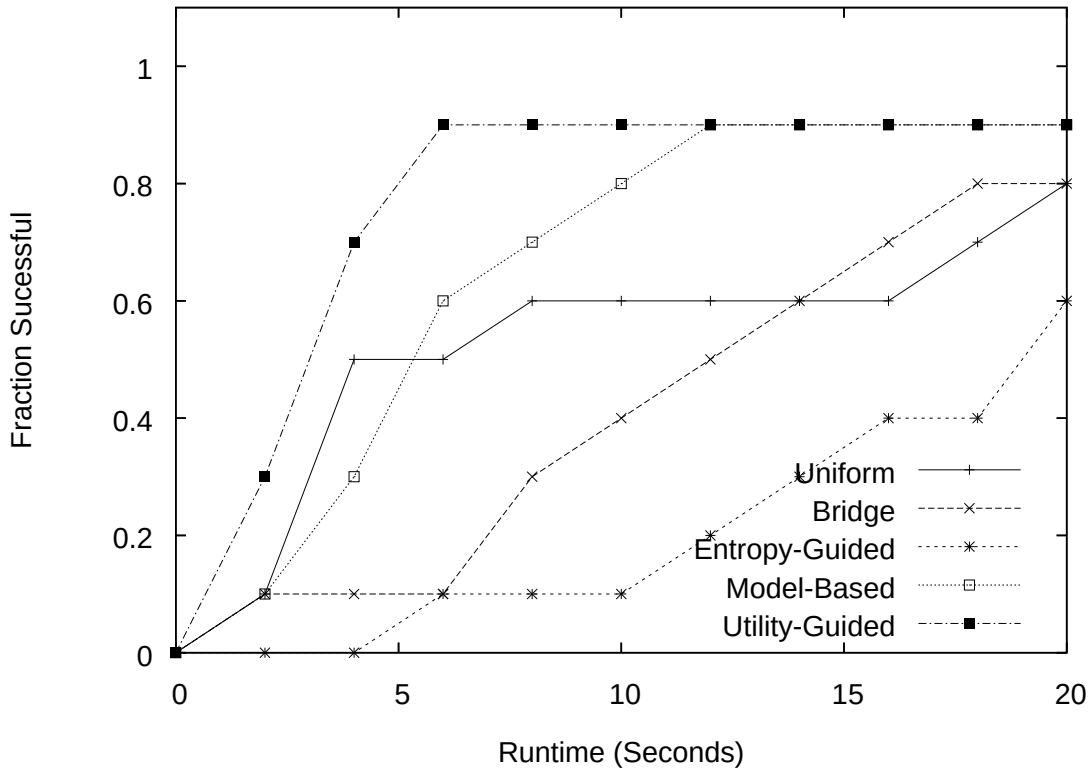
(a) 4-DOF mobile manipulator



(b) 6-DOF mobile manipulator



(c) 9-DOF arm



(d) 12-DOF arm

Figure 5.18. Comparing coverage performance of different parts of utility-guided planning and other planning algorithms

and experimental domain we ran ten different coverage experiments with different random seeds. The averages for the ten runs in each workspace are shown in Figure 5.18.

In the both sets of graphs, the comparison of the performance of utility-guided sampling with entropy-guided [19] and model-based [21] sampling is important. Entropy-guided sampling uses roadmap structure, while model-based sampling uses an approximation of the state of the configuration space. Each of these approaches does well in one environment and poorly in the other. The performance of each sampler is dependent on the presence of suitable environmental features. Model-based sampling does poorly in the mobile robot environment because it is uniformly interested in the wall, not just in the opening that is crucial for the solution of the problem. On the other hand, entropy-guided sampling does poorly in the fixed arm world where its greedy, direct attempts to connect disjoint configuration space components generally end in failure. By blending together both sources of configuration space information the utility-guided sampling strategy achieves high performance in both environments.

An important consideration for motion-planning is sensitivity to degenerate behavior in particular types of configuration space. Utility-guided planning avoids the degenerate behavior exhibited by its constituent components: entropy-guided and model-based sampling. The complementary combination of two sources of information by utility-guided sampling makes the resulting planner significantly more robust.

The results of these experiments also reinforce the results of the artificial and real world experiments presented previously. The use of the utility-guided sampling strategy improves the performance of the PRM algorithm. In all cases the utility-guided strategy reduces the average runtime by at least a factor of two.

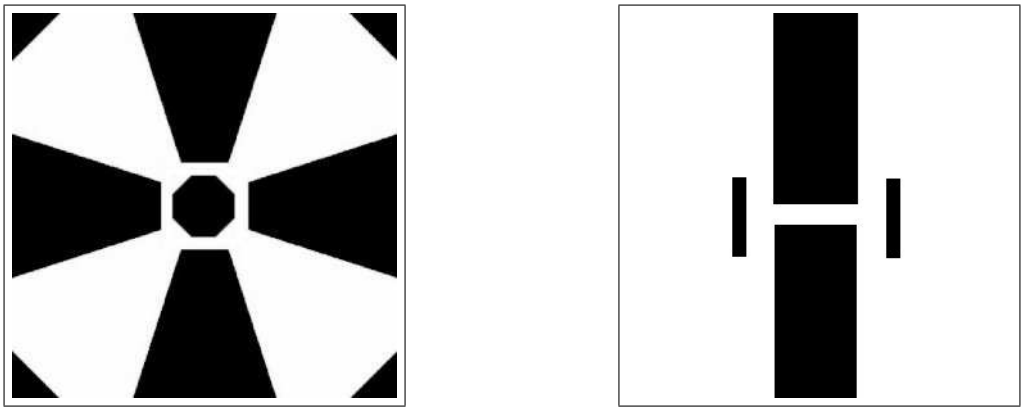
5.6.5 Comparison of heuristic and actual utility-guided sampling

The exact calculation of the expected utility for every configuration in a configuration space requires computation that is exponential in the dimensionality of the configuration space. Consequently, the practical implementation of utility-guided sampling uses a heuristic approach which attempts to select samples with maximal expected utility using significantly less computation. An important issue is understanding how closely the heuristic models a true utility-guided sampler.

To achieve this comparison, we implemented a non-heuristic version of utility-guided sampling. This non-heuristic sampler iterates through all possible configurations in the configuration space and calculates each sample's expected utility using the formulas described in Section 5.3. We compared the performance of this non-heuristic sampler with the heuristic utility-guided sampler used in previous experiments.

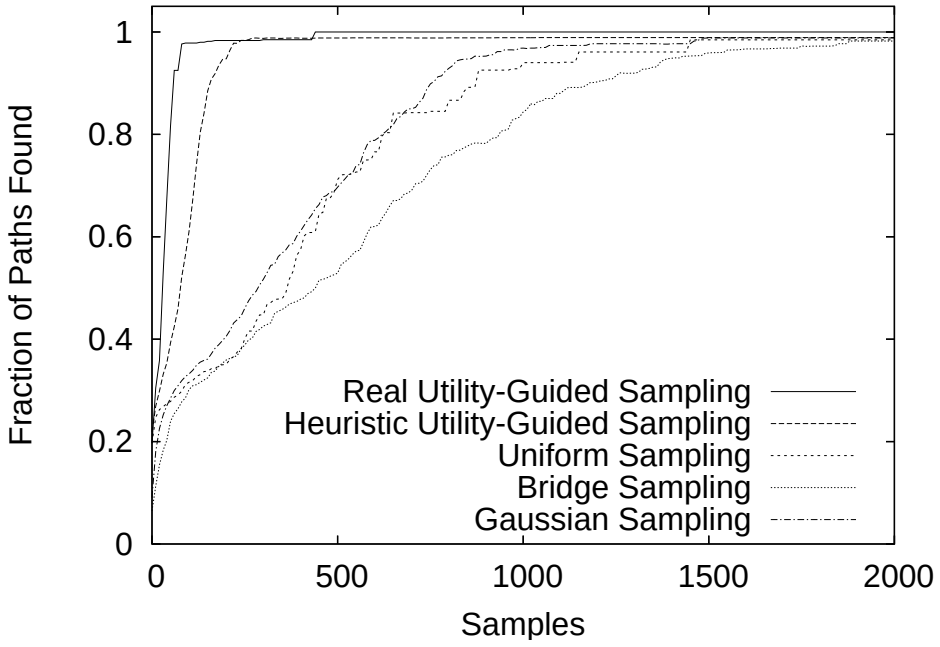
Because the non-heuristic sampler is roughly 100 times slower than the heuristic sampler, the comparison of the two samplers was done as a function of the number of samples selected rather than the runtime of a planner using each sampler. Experiments were run in two workspaces with a two degree-of-freedom point robot. These workspaces are shown in Figure 5.19. Two-dimensional configuration spaces were used in order to make the experiment computationally feasible. For each workspace, coverage experiments, similar to those in Section 5.6.3, were run. Each planner incrementally stopped after selecting a number of samples. Each planner was then asked a series of one hundred random path queries which were all known to have solutions. The fraction of the queries which the particular planner could successfully answer was recorded. The graphs in Figure 5.20 show these results averaged over fifty different trials with different random seeds.

These results demonstrate that heuristic utility-guided sampling does a good job of modeling actual utility-guided sampling. As expected, the non-heuristic utility-guided sampler outperforms heuristic utility-guided sampling. However, the shape

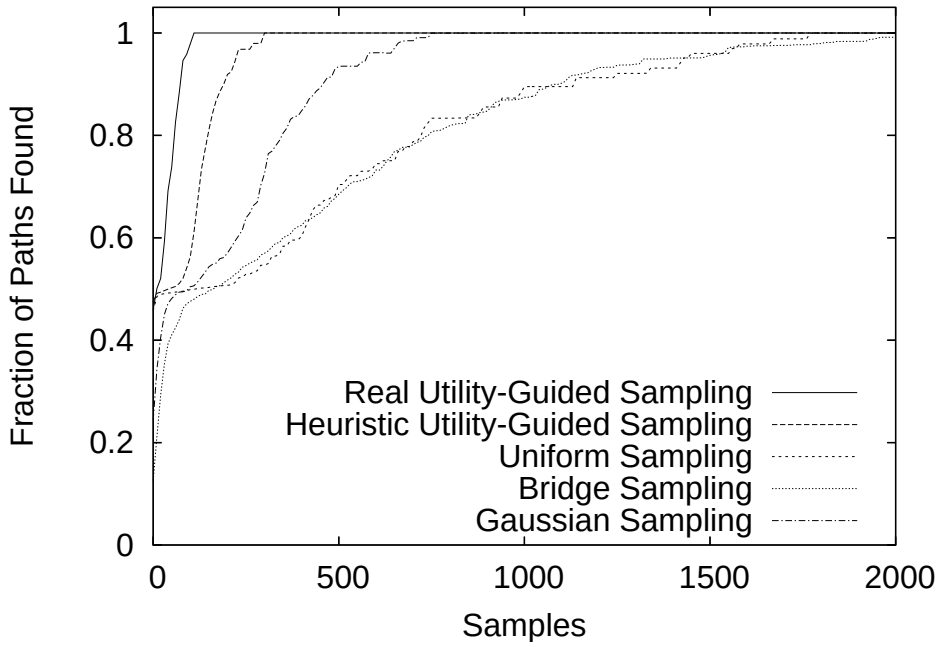


(a) Four compartment workspace (b) Passage workspace

Figure 5.19. Workspaces used for comparing heuristic and actual utility-guided sampling



(a) Four compartment workspace



(b) Passage workspace

Figure 5.20. Comparison of actual utility-guided sampling with heuristic utility-guided sampling, bridge sampling and uniform sampling in two different worlds.

of heuristic utility-guided sampling's performance curve closely mimics that of real utility-guided sampling. Additionally, the graphs provide further corroboration of earlier results which demonstrate that utility-guided sampling, either heuristic or not, significantly outperforms other guided sampling strategies as well as uniform sampling.

Chapter 6
Single-Query Planning

6.1 Introduction

In the real-world, a robot is always in some particular state. Consequently, the relevant planning question is: What movements *from this current state* are possible? or alternatively: How can I move from my current state to some specific goal state? Rather than using a multi-query algorithm to compute all possible configuration-space paths, these questions can be more efficiently answered by examining the connectivity of the configuration space in the region surrounding the current state of the robot. More generally these planning algorithms compute the evolution of the set of paths originating at a particular state. These planners operate by diffusing a connectivity tree through the configuration space. Rather than constructing a roadmap graph with equal density across the entire configuration space. The so-called "random-tree" is rooted at a particular configuration and expands to approximate configuration space connectivity. Random-trees are grown by sampling collision-free trajectories through the configuration space. These trajectories form the new branches in the random tree. Despite a bias toward a particular region of configuration space, given sufficient expansion the random tree will eventually diffuse across the entire configuration space and approximate its entire connectivity.

Random tree expansion is a successful algorithmic approach to sampling-based single-query motion planning. A single-query planning problem is defined by two points in the state (or configuration) space of a robot. A single-query planner attempts to connect these states by a sequence of valid transitions through state

space. The success of random tree expansion planners in the single-query planning domain can be explained by two advantageous characteristics of random tree expansion: *1)* random trees rapidly explore state space, leading to an effective search for a solution to the planning problem, and *2)* the exploration of state space can be performed based on control inputs, making the algorithm applicable to systems that are subject to differential constraints.

The important question in random tree planning is how to expand a tree through the state space. In this chapter I present a random-tree planner which adapts its expansion to its current state. Experimental results indicate that this online, adaptive approach leads to significant performance improvements relative to previous tree-based planners. These improvements become more pronounced as the dimensionality of the state space increases.

To achieve these performance improvements, I first present a compressive algorithmic framework for tree-based single-query planning. The framework generalizes existing tree-based planners, which can be seen as instantiations of the framework. It also reveals the fundamental parameters that govern the expansion behavior of random trees. I propose a utility-guided approach to controlling expansion behavior by adjusting these expansion parameters throughout the planning process. Given the state space information obtained in previous expansion steps of the planner, the approach maximizes the expected utility of the next expansion step. The resulting planning framework directs state space exploration so that—given the available information—maximum expected progress towards a solution is made.

6.2 Exploration With Random Trees

Tree-based single-query motion planners are well suited to solving connectivity problems in high-dimensional spaces. These planners grow trees of state transitions representing spatial connectivity. Each tree is rooted at an initial state. Subsequent

growth via sampling may add a branch to any node in the tree. A path in this tree represents a valid sequence of state transitions through state space.

To facilitate the analysis of random-tree planning I begin by distilling the general characteristics of single-tree expansion into a generic algorithm. Next, the planner is extended to consider an arbitrary number of different random trees. These generalized algorithms expose the individual modular components that compose any tree-based planning algorithm.

6.2.1 Generalized Random-Tree Expansion

Every random tree begins with a single state. This tree is expanded through sampled state transitions so that it begins to approximate the configuration space connectivity. The goal of each tree expansion is to explore the connectivity of the state-space with a bias toward one or more local regions.

Exploratory growth initially selects a node in the random tree as the origin of the new tree branch. Once a node has been selected, its location is used to select an exploratory direction for the expansion. This direction may be a simple linear path through configuration space or it may be a more complex movement through state space. Once a node and direction have been selected, the length of the growth is chosen. When the expansion is completely instantiated it is validated If the state transition is collision-free, the tree is expanded to include the transition. The process of exploratory growth continues until some stopping criterion is achieved. The resulting tree approximates the connectivity of the state-space with a bias toward the region surrounding the root of the tree. The general algorithm for tree expansion is given in pseudo code in Figure 6.1.

This basic tree-based exploration algorithm identifies four algorithmic components that determine the expansion behavior of the trees generated by the planner: node selection, direction selection, distance selection, and connection. All existing planners

RandomTreePlanner(q_s)
1: $T = Tree(q_s)$
2: **while** (stopping criterion not met)
3: q_n = SelectNode(T)
4: $\vec{v}$ = SelectTrajectory(T, q_n)
5: d = SelectLength($T, q_n, \vec{v}$)
6: Extend($T_a, q_n, \vec{v}, d$)

Figure 6.1. The generalized random tree expansion algorithm

can be described by choosing particular implementations for these components. These implementation choices are summarized in Table 6.1 summarizes for several existing tree-based planners.

6.2.2 Generalized Random Forest Motion Planning

Previous random-tree planners solve the single-query motion planning problem. Consequently, they do not consider the expansion of a single tree, but rather a pair of trees. Bi-directional tree-based planners solve the single-query problem by growing two configuration space trees, one rooted at the start configuration and one rooted at the goal. The trees are alternately grown until they are connected to each other. Tree growth occurs in one of two different modes: connecting growth or exploratory growth. Connecting growth attempts to connect the two trees together. Exploratory growth attempts to expand a tree's representation of configuration space connectivity. The focus on the single-query problem means that most approaches have only considered planning with two random-trees. Such approaches can be generalized to forests of arbitrary size. For example, a planner may wish to compute a path from the robot's current position to any number of equally satisfactory goal locations. Alternatively, trees may be rooted using workspace information [116]. The following describes a generalized random-forest motion planner that coordinates the expansion and connection of arbitrary numbers of individual random-trees.

```
RandomForestPlanner(q_1 ... q_n)
1:    T = {Tree(q_1) ... Tree(q_n)}
2:    while ( stopping criterion not met )
3:       (T_a, T_b) = SelectTrees(T)
4:       if(Connect(T_a, T_b))
5:          T = T - T_a
6:          T = T - T_b
7:          T = T ∪(T_a + T_b)
8:       else
9:          if(shouldExpand?(T_a))
10:            Expand(T_a)
11:         if(shouldExpand?(T_b))
12:            Expand(T_b)
```

Figure 6.2. The generalized random forest planning algorithm

The operation of random-forest planner is simple. It begins by constructing a set of trees rooted at each of the configurations it receives as parameters. Once the trees are constructed, it selects a pair of trees for a connection attempt. Next the planner attempts to connect these trees via a collision-free state-space trajectory. Several different algorithms for connecting growth have been proposed [64, 107]. Regardless of the implementation, connecting growth attempts to use the current representation of configuration space connectivity to identify a successful path. If the connection is successful, the trees are merged. If this connection is not successful, the planner decides whether or not to expand each of the random trees so that future connection attempts may be feasible. This process of selection, connection and optional expansion repeats until some stopping criterion is reached. Pseudo-code of this general random-forest planning algorithm is given in Figure 6.2.

6.2.3 Related Implementations of Random-Tree Planning

The performance of a single-query planner is defined by the speed with which it can discover a collision-free path connecting two specified configurations. Achieving

maximal efficiency requires careful implementations of each function in the generalized planning algorithm. There has been considerable research into tree based planning that provide different implementations for some or all of these functions. A summary of these implementations is shown in Table 6.1.

When viewed through the general random-tree algorithm, comparisons between existing tree-based methods are simplified. Through this comparison it can be seen that the largest amount of research has explored the selection of the node to expand and to a lesser extent, the direction of this expansion. Relatively little research has examined the role of the exploration length or the algorithm for connecting growth. Interestingly, the one approach that examines connecting growth [107] does not use exploratory expansion. In the following section I propose an application of the general utility-guided planning framework to random tree expansion.

6.3 Utility-Guided RRTs

The generalized algorithm for random-tree planning identifies a set of modular components that make up a random-tree planner. The implementation of all of these modules affects the performance of the planner. The following first considers the application of the utility-guided framework to random tree expansion. Subsequently it considers utility-guided random-forest planning.

6.3.1 Utility of node expansion

The first step in random-tree expansion selects the node to be expanded. The utility of selecting a particular node is tied to the possibilities for meaningful exploration of configuration space connectivity still possible from that node. Of course, without a precise understanding of the state of the configuration space surrounding a node this is impossible to know. I estimate this quantity to be inversely proportional to the number of exploration attempts originating from the node.

Algorithm	Node Selection	Expl. Direction	Expl. Length	Connection	# Trees
Z^3 [9]	Uniform Random	Local Planner	Local Planner	N/A	1
RRT-Connect [64]	Voronoi Bias (V.B.)	Voronoi Bias	Constant	Nearest Node	2
DD-RRT [46]	DD V.B.	DD-V.B.	Constant	N/A	1
ADD-RRT [117]	ADD V.B.	ADD-V.B.	Constant	Nearest Node	2
OB-RRT	Voronoi Bias	Hybrid	Hybrid	Nearest Node	2
Blossom-RRT [87]	Voronoi Bias	All	All	N/A	1
SBL [93]	C-Space Density	Uniform Random	Shrinking Neighborhood	Bridge Node	2
Exp. Spaces [40]	Tree density	Tree density	$<$ Constant	$<$ Constant	2
Guided Exp. Spaces [83]	Heuristic	Heuristic	$<$ Constant	$<$ Constant	2
Adapt. Single-Query [107]	none	none	none	Heuristic	2
RRFT [54]	Dyn. or Hist. Based	Adapt. Goal Biased	$<$ Constant	N/A	1

Table 6.1. Summary of tree-based planners presented in the literature

$$\text{Utility}(q) = \frac{1}{\#\text{ Previous Expansions}}$$

This estimator predicts high utility for nodes on the fringe of the tree and a uniform, lower utility for interior nodes that are far removed from the leaves. By choosing other estimators for this utility, it is possible to change the balance between exploration and refinement during three growth. An additional benefit of this node selection strategy is that it can be implemented in constant time, avoiding the expensive nearest-neighbor queries used by other approaches such as the Voronoi bias.

6.3.2 Utility of expansion direction

Once a node has been selected, a direction for expansion must be chosen. Because the goal of this expansion is exploration, the most useful directions for exploration are those furthest from previous exploration originating from the node. Let the set D be the set of previous expansion directions originating from some node q. The utility of of some new expansion direction d' is given by the formula:

$$\text{Utility}(q, d') = \sum_{d_i \in D} -\text{success}(d_i) cos(\theta(d', d_i))$$

In this formulation, θ is the angle between the two vectors. Because the directions are unit vectors, this can be calculated from their dot product. The value success($\vec{d_i}$) is used to bias selection towards directions that are likely to be unobstructed. In the implementation this function returns a constant c for directions $\vec{d}$ that resulted in a successful expansion, and $c/2$ otherwise. Based on this function, the planner uses past experience to avoid subsequent expansions that are likely to be obstructed.

6.3.3 Utility of exploration length

The utility estimators for node selection and direction selection do not rely on estimates of probability. The probability in both cases is always one, since every node

selection and every direction selection will always be successful. The remaining two components of the algorithm, exploration distance and connection attempt, traverse the state space and therefore have to consider the probability of this traversal being successful. A traversal that is unlikely to be successful has low expected utility.

The remaining two components of the utility-guided algorithm, namely the exploration step during tree expansion and the attempt to connect the two trees, are performed in a very similar fashion. However, there is one important distinction: during exploration the exploration direction is chosen as described in Section 6.3.2, whereas the expansion direction during the connection attempt is given by the state of the other tree. I begin by describing the process of determining the exploration distance.

At this point, the algorithm has selected a node q and a direction $\vec{d}$ for the expansion. I now need to determine a distance δ for this expansion. Assuming that δ has been determined, the planner then perform collision checks in increments of ϵ along the expansion direction to validate that the expansion is valid, until the distance δ. The planner could determine δ based on the information available at this point of the planning process. However, it would be advantageous to determine it incrementally by exploring the state space along $\vec{d}$. This will allows us to gather additional information about the state space in the expansion direction, leading to improved estimates of utility.

To determine δ, I proceed in increments of $alpha >> \epsilon$ along $\vec{d}$. At each α increment, I evaluate the expected utility of that point a'. If this expected utility exceeds a threshold $u_{\min}$, the algorithm decides to expand to that point and δ is temporarily set to the distance between q and q'. The planner now validates the connection between q and q' in ϵ increments. This validation obtains more information about local state space features that help to estimate the utility at the next α increment along $\vec{d}$. This process continues until the utility of q' is below the threshold or the

UtilityExtend(T, q_s, q_e, M)

```
1:    q_i = q_s
2:    do
3:       q'_i = q_i + α
4:       if(Utility(q_i → q'_i) < μ)
5:         break
6:       if (Free(q_i → q'_i))
7:         AddToTree(T, q'_i)
8:       q_i = q'_i
9:    while (q_i != q_e)
```

Figure 6.3. The utility-guided extension algorithm used for both exploration and connection

connection between two states is invalid. Figure 6.3 shows this extension algorithm in pseudo-code.

The planner estimates the expected utility of a particular point q' based on the probability of it being collision free and its utility. The former can be obtained from the state space model described in Section 5.4. To determine the utility of a point n', I consider its distance δ to q, the node being expanded. If this distance is larger than a threshold $\delta_{\max}$, the utility of q' is zero, otherwise it is identical to the distance δ itself. The introduction of this cut-off is motivated in Figure 6.3.3. The exploration step of the algorithm is intended to move out of the shadow of state space obstructions so that the connection step (described next) has an increased chance of successfully connecting the two trees. If the exploration length is too large, the likelihood of connecting the two trees is not improved and the cost of validating the transition becomes very large.

Other researchers have also identified excessively long exploration steps as problematic for several single-query planners [64, 14], thus providing support for the choice of utility function.

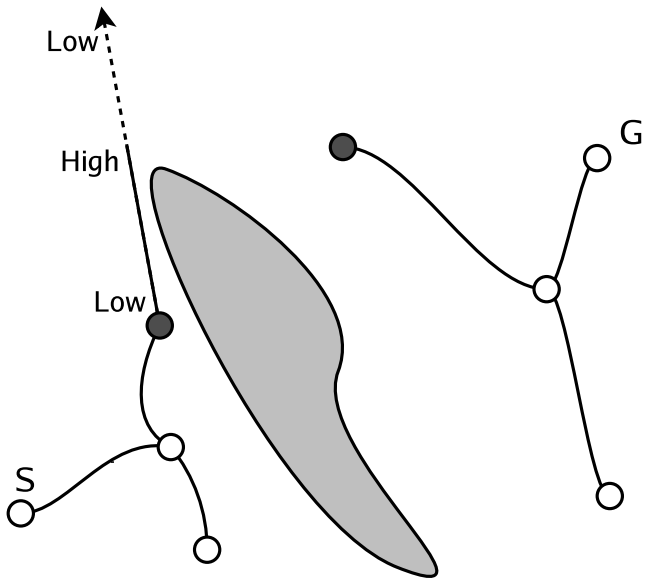


Figure 6.4. An illustration of utility for expansion distance. Expansion beyond the shadow of the obstacle does not increase the utility of the expansion.

6.3.4 Selecting trees for connection

When a tree has been successfully expanded, the new growth may make it possible to connect two trees together into a single tree. When there are only two trees present, there is no choice as to which trees to connect. However, in general random-forest planning there are many different pairs of trees that could be connected. The selection of trees for a connection attempt has significant impact on the performance of the planner. When successful, connecting two trees significantly improves the planner's approximation of configuration space connectivity. However, indiscriminate connection attempts between trees often leads to repeated failed connections that waste significant amounts of computation. Fortunately, by combining the utility of connecting the trees with the probability that such a connection is possible into the expected utility of the connection, a pair with maximal expected benefit to the planner can be selected.

The utility of connecting two trees is largely identical to the utility of connecting two disjoint connected components in the roadmap graph discussed in other work [19].

Consequently, roadmap entropy and information gain again provide utility function. The probability that two trees can be connected is estimated from previous connection attempts and expansions of the trees. If a previous connection attempt failed and neither tree has subsequently been expanded, the probability of connection is zero. Otherwise, the probability is proportional to the number of expansions of the trees. If no previous connection has been attempted the probability is optimistically set to one.

$$\text{Utility}_e(T_a, T_b) = \begin{cases} 0 & \text{if no expansion since failed connection;} \\ \propto (\text{Expds.}(T_a) + \text{Expds.}(T_b)) \\ \quad \text{Utility}(T_a + T_b) & \text{if Connections}(T_a, T_b) > 0; \\ \text{Utility}(T_a + T_b) & \text{if Connections}(T_a, T_b) = 0; \end{cases}$$

6.3.5 Utility of Connecting Growth

Attempting to connect the two trees proceeds in a similar fashion as the determination of the expansion length. Now, the state q' obtained by the expansion step itself is used for expansion. The expansion direction $\vec{d}$ is given by the direction towards the closest node of the other tree. The definition of utility does not include the cut-off, since the connection step attempts to connect the trees in a greedy fashion. The expected utility of an increment in the connection attempt can only become smaller than the minimum threshold $u_{\min}$, if the probability of n' being collision-free becomes very small. Effectively, the connection step proceeds in a greedy fashion with an α look-ahead based on the state space model.

Unfortunately, greedy search often leads to pathological behavior when the greedy path diverges significantly from the true path. In motion planning this causes planners to repeatedly attempt obstructed paths that are preferred by the greedy planner. This problem has been identified as a hazard for several single-query planners [64, 14].

Algorithm	Node Select.	Expl. Direction	Expl. Length	Connection	# Trees
Utility-RRT	Utility	Utility	Utility	Utility	2
Vor./Util. RRT	Voronoi Bias	Voronoi Bias	Util. Exp.	Util. Exp.	2

Table 6.2. The two utility-guided RRT algorithms described in terms of the general tree planning algorithms

Fortunately a planner that is greedy with respect to *expected* utility generally avoids pathologic behavior. The expected utility calculation considers the probability that the expansion is obstructed as well as the utility of the expansion. Since obstructed expansions have no utility, pathological expansions through obstructed configuration space regions are not chosen. If, because the predictive model is incorrect, an obstructed expansion is chosen, information from that obstructed expansion is incorporated into the predictive model improve its predictions of future expansions.

6.3.6 Utility-Guided RRT Planning

The utility-functions presented previously are used to instantiate a utility-guided implementation of the general algorithm presented in Section 6.2. To compare the role of various utility-guided exploration choices, two different utility-guided algorithms are used. One is a hybrid algorithm that uses the traditional Voronoi bias for selecting nodes and exploration direction and utility-guided expansion distance and connection attempts. The other algorithm uses utility to guide all aspects of random-tree expansion. In the context of the Table 6.1, the two utility-guided random-tree algorithms are summarized in Table 6.2.

6.4 Experiments

The goal of random-forest planning is to quickly identify a collision free paths. Through the generalization of the random-tree algorithm I have identified several components in this planning process that affect the efficiency and reliability of the planner that have previously been unexplored. The following presents planning ex-

```
Collision Check(q)
1:      d = distanceToOrigin(q)
2:      if(d > sphereRadius+depth)
3:        return false
4:      if (d < sphereRadius)
5:        if(q[0] < 0)
6:          return false
7:        d = distanceToXAxis(q)
8:        return (d > holeRadius && d < holeRadius+depth)
9:      d = distanceToXAxis(q)
10:     return (dist > holeRadius)
```

Figure 6.5. The arbitrary dimension bugtrap collision checker

periments that compare the performance of the utility-guided random-tree planner with adaptive dynamic-domain RRT [117] a state of the art planner. In these experiments, the utility-guided planners are limited to planning with only two trees to make the comparison with the bi-directional dynamic-domain planner fair.

6.4.1 Bugtrap experiments

Recently, researchers [117, 46] have identified the so-called "bugtrap" (Figure 6.8) as a challenging configuration space feature for traditional RRT methods. These previous experiments have only considered bugtraps in two-dimensional configuration spaces. Because challenging motion planning problems occur in higher dimensional configuration spaces, it is unclear that two-dimensional worlds offer much insight into the performance of motion planning algorithms in typical configuration spaces. Defining bugtrap configuration spaces capable of arbitrary dimension ensures a more complete comparison. This bugtrap consists of a single hyper-sphere obstacle with some thickness. The hyper-sphere is pierced by a hyper-cylinder of the same thickness oriented along the x axis and extending from the origin to the edge of the hyper-sphere. If a configuration is within this shell it is considered obstructed, otherwise it is free. The collision check test for the arbitrary dimension bugtrap is shown in Figure 6.5.

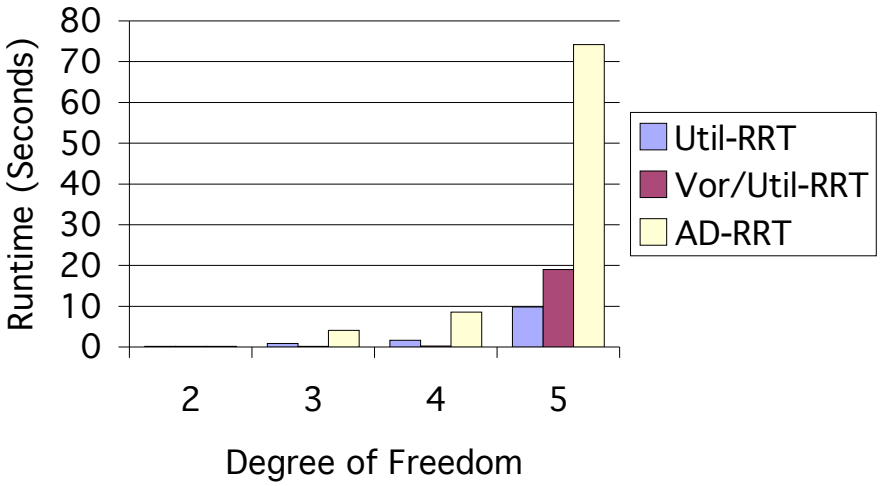


Figure 6.6. Results comparing planning algorithms for bugtrap worlds of varying dimensionality

DOF	A.D. RRT	Hybrid Voronoi/Util. RRT	Full Utility RRT
2	0.14 (0.21)	0.14 (0.39)	0.17 (0.41)
3	4.06 (9.26)	0.23 (0.23)	0.83 (1.52)
4	8.56 (14.20)	0.37 (0.38)	1.63 (4.12)
5	74.17 (163.92)	19.01 (31.73)	9.79 (21.24)

Table 6.3. The average runtime, with standard deviations in parenthesis, for adaptive domain RRT and utility guided RRT for bugtrap worlds of varying dimension

I performed planning experiments in bugtrap worlds with two, three, four and five dimensions. In each world, each planner was asked to compute a path between a random point inside the bugtrap to a random point outside the bugtrap. The length of time to compute this path was recorded. The average results for 50 path queries are shown graphically in Figure 6.6. They are also shown numerically in Table 6.3.

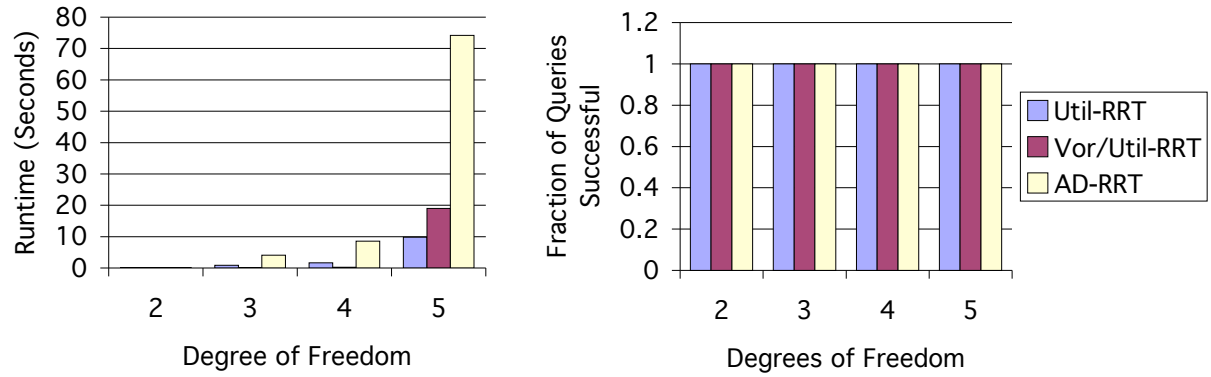
From these results it can be seen that as the dimensionality of the bugtrap increases, utility-guided random-tree expansion outperforms adaptive-domain expansion. This shows that utility-guided exploration is choosing expansions for the trees which more quickly identify the successful path out of the bugtrap. Because utility-

guided planning uses its past experience to guide tree expansion, exploration is not wasted on paths through the bugtrap's walls that are likely to be obstructed.

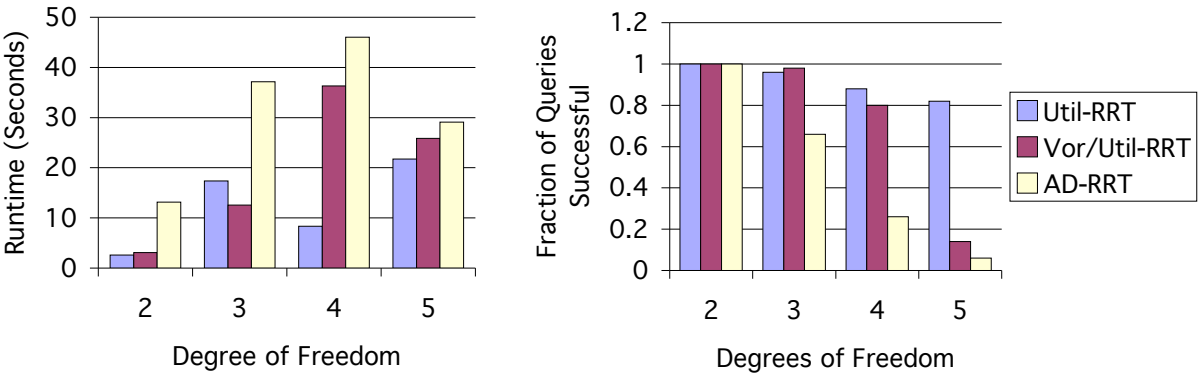
In the previous experiments, the configuration space was only slightly large than the radius of the hyper-sphere defining the exterior of the bugtrap. Experiments also investigate the performance of random-tree planners when the size of the configuration space relative to the bugtrap varied. For these experiments, the size of the bugtrap is constant, but the size of configuration space is varied. Results showing the performance of all planners in different sized configuration spaces of varying dimensionality are shown in Figure 6.7. In these experiments, all planners were allowed to run for a maximum of five minutes. If the planner failed to provide a motion plan after five minutes, it was killed. The left-hand graphs report the average runtime of successful motion plans. The right-hand graphs show the fraction of path planning attempts that succeeded for different sizes and dimensions. All graphs represent the average of fifty path queries.

The results with different ratios of bugtrap size to configuration space size demonstrate the efficiency of utility-guided planning. In general, for all of the planning algorithms, as the dimensionality of the configuration space increases, the runtime increases as well. However, the runtime of the complete utility guided algorithm increases at a significantly slower rate. Runtime also increases as the size of the bugtrap decreases relative to the size of the configurations space. Again, this increase is significantly smaller for the complete utility guided algorithm than for adaptive domain random-tree planning. The runtime of the hybrid Voronoi/utility-guided algorithm generally falls between the other two algorithms.

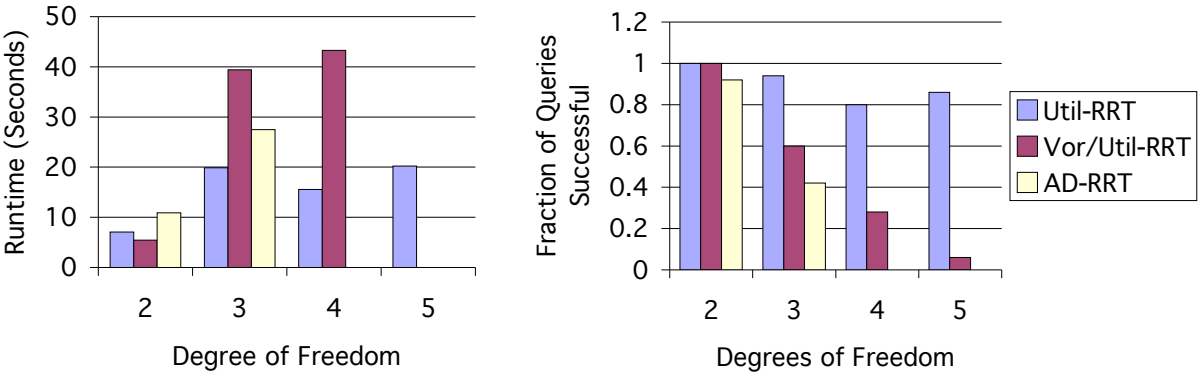
The results also demonstrate the reliability of utility-guided random tree planning. When the size of the bugtrap is large relative to the configuration space, all of the algorithms are completely successful at finding paths. However as the size of the bugtrap decreases relative to the size of the configuration space and the dimension-



(a) Small Configuration Space



(b) Medium Configuration Space



(c) Large Configuration Space

Figure 6.7. Average planner runtime for different ratios of configuration to bugtrap size in configuration spaces of varying dimensionality

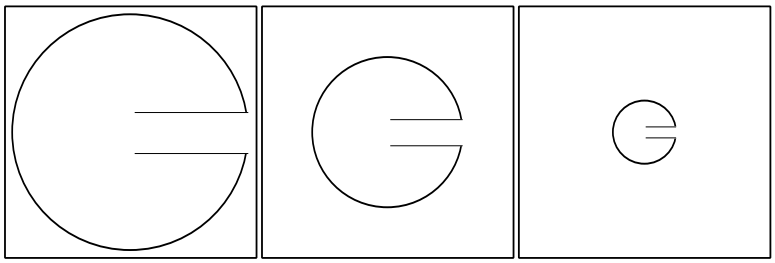


Figure 6.8. From left to right bugtraps in small, medium and large configuration spaces

ality of the bugtrap increases, the reliability of the planners decreases substantially. However, in all cases the complete utility-guided planner is significantly more robust than either adaptive-domain planning or the hybrid Voronoi/utility-guided planner. In all cases the complete utility-guided planner never falls below an 80% success rate, while the other two algorithms drop to below 10% success. These results demonstrate that the utility-guided random tree planner is both more efficient and more reliable. These attributes are necessary for effective real-world motion planning.

6.4.2 Real World Experiments

The experiments in bugtrap worlds demonstrate that utility-guided random-tree planning can improve performance in challenging configuration spaces. However, from a practical perspective it is important that the algorithm improve the performance of real-world robotic planning as well. Thus I examined planner performance in the context of two related real-world tasks for a 14-DOF humanoid torso (Figure 6.9). Both tasks examine the assembly of pipes in an obstructed environment. These experiments correspond with the overall goal of this humanoid platform performing assembly and service tasks on the exterior of the international space station. The motion planning problems consist of a start position with the arms extended on the outside of the box where the assembly will take place. The goal position is inside the box with the pipes oriented and ready for a lower-level force-controller to orchestrate

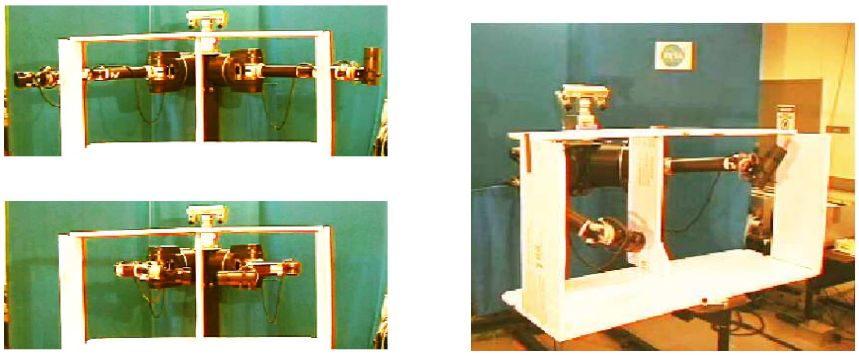


Figure 6.9. The humanoid robot in its start and goal configurations (left) and enacting a successful motion plan (right)

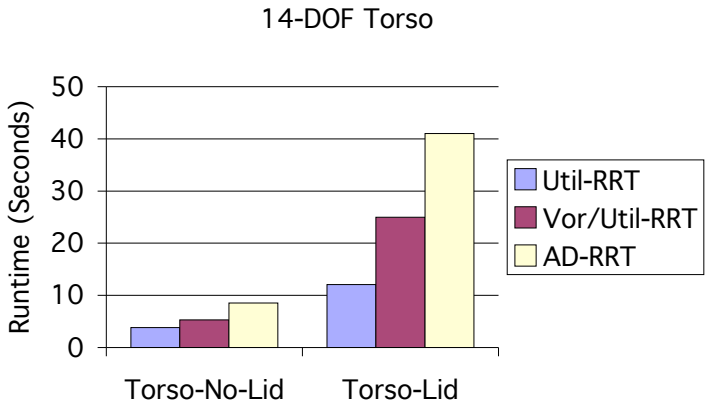


Figure 6.10. Average runtime of various planning algorithms for the 14-DOF humanoid torso in two different environments

the final assembly. To experiment with the motion planner on problems of varying difficulty, I performed experiments with and without a lid on the box. Images of the start and goal for both of these experiments are shown in Figure 6.9. For each of these experiments I performed 50 different requests for the same path query to planners seeded with different random seeds. The average runtime and standard deviation for each planner and each task are given in Table 6.4.2.

These results show that utility-guided planning is well suited to real-world motion planning. The hybrid planner provides nearly a two times speed up over existing state

World	A. D. RRT	Hybrid Voronoi/Util. RRT	Full Utility RRT
Box w/o Lid	8.54 (4.54)	5.32 (2.77)	3.81 (1.98)
Box w/ Lid	41.01 (22.05)	24.97 (16.44)	12.05 (8.08)

Table 6.4. The average runtime, with standard deviations in parenthesis, for adaptive domain RRT and utility guided RRT for two related real-world environments

of the art techniques, while the complete utility-guided planner provides between a two and four times speed up. The results are especially encouraging because the performance improvement of complete utility-guided planning increases as the complexity of the problem increases. This demonstrates that in addition to improving runtime performance, utility-guided planning scales better with respect to the complexity of the underlying problem. Also of significance are the runtimes themselves. In a moderately complex world (the box without the lid) utility-guided planning has an average runtime that is adequate for planning in slowly changing dynamic environments. This demonstrates that the utility-guided algorithm is capable of real-world motion planning even when objects in the environment are in motion.

Chapter 7
Planning Under Uncertainty

7.1 Introduction

In most real-world robotics, the only source of information about the workspace are sensors such as cameras and range-finders. Any representation of the workspace constructed from sensors is subject to noise and error in their observations. Consequently, there is uncertainty about the true location of obstacles in the workspace. This problem is illustrated in Figure 7.5, which shows the uncertainty perception of a simulated office environment. Uncertainty in obstacle location has significant implications for real-world motion-planning algorithms. A planner may compute what is thought to be a collision-free path, only to discover it is obstructed when it is physically executed. Or, a planner may believe no paths are feasible when paths in fact exist. A planner must understand uncertainty in its perception to compute paths that minimize the expected costs of such planning failures.

Sampling-based algorithms [52, 64] have dramatically improved the speed of state-of-the-art motion planners. However, nearly all of these algorithms assume that when a collision check is performed, its results are guaranteed to be correct. Collision checks which use uncertain perception of the environment do not provide this guarantee. Instead, they output a probability that their prediction of the configuration's state is correct. In this work I present a new roadmap-based planning algorithm which identifies configuration space paths which minimize the risks associated with uncertainty. Experimental results indicate that it computes more robust paths than existing sampling-based planners that ignore uncertainty.

To achieve this improvement, the algorithm integrates knowledge of sensor uncertainty directly into its planning. My planner models expected sensor error to estimate the probability that each edge in the roadmap is unobstructed despite any mis-perception. Edge probabilities are used to estimate the probability a path is unobstructed. The probability of each path is used to select the path with the greatest *expected utility* for physical execution. Expected utility is a weighted combination of the benefit (positive utility) of the path being free and the cost (negative utility) of the path being obstructed when physically executed.

Often the initial information available to the planner is insufficient for computing a acceptable path. If all paths are highly uncertain or the cost of failure is high, the expected utility of a path may be negative. In this case, additional sensing is needed to reduce perceptual uncertainty. In turn, this new information provides an opportunity to re-plan and possibly identify a new path with maximal expected utility. By iterating the invocation of this planner from within a "sense, plan, sense, act" control loop, it is possible to generate robust motion for complex robots in uncertain environments.

In principle, such a control loop could use any planner. However, the explicit consideration of uncertainty and sensing actions in the planning process has several advantages. First, it allows the planner to operate despite inaccurate knowledge of the environment. Second, dynamic environments can be addressed simply by increasing the uncertainty in outdated regions of the environment. Third, the consideration of uncertainty enables the planner to determine motions that have a high probability of success in the absence of perfect sensing. Finally, the generation of on-board or off-board (smart room) sensing actions based on the particular motion objective minimizes the cost of sensing required to obtain the information required for successful planning.

My experimental evaluation demonstrates the proposed planner's can successfully plan using partial and uncertain information about the world, shows that the explicit consideration of uncertainty can improve the success rate of plans, and validates that guided exploration is effective at reducing the sensing necessary for planning.

In real-world robotics, errors in perception introduce errors in the workspace representation a planner uses to compute feasible motions. Given the possibility of error due to uncertainty and the severity of the potential consequences of this error, it is important for a planner to reason about such potential mistakes during planning.

In real-world planning a robot begins with inaccurate and/or incomplete information about its workspace. When motion is necessary, it must use this limited information to identify the best possible path available. The formalization of Bernoullian utility [47] is used to rank paths. Every successful path has a benefit, or positive utility, which is a function of its length, kinematic properties or any other task-specific features. Likewise, every failed path has a *cost*, or negative utility. Cost is computed from the consequences of a path failure such as a physical collision, or simply failing to achieve a goal. Expected utility combines positive and negative utility together weighted by the probability that the path is unobstructed. This probability estimate is derived from an understanding of how errors occur in the representation of the workspace. The expected utility of some path P is the summation over its constituent edges p_i:

$$\sum_{p_i \in P} (1 - P(\text{Err.}|p_i))\text{Utility}(p_i) + P(\text{Err}|p_i)\text{Cost}(p_i).$$

The path with greatest expected utility maximizes the rewards and minimizes the risks of physically enacting a particular plan.

For a variety of reasons, the initial information about the workspace available to the robot may be insufficient to identify an acceptable path. When uncertainty is

great or the cost of failure high, the expected utility of the best available path may be negative. In such cases, a planner's only option is to use additional sensing to explore the world and refine its representation of the workspace. Because sensing the world is expensive, information from the previous planning is used to guide further sensing to areas relevant to the particular path query. This sensing reduces uncertainty and provides additional information about the workspace. Using this new information, the planner can re-plan a new path with maximal expected utility. Planning and sensing alternate until a satisfactory path is identified and sent to the robot for execution. Although this path has positive expected utility, when executed it may turn out to be obstructed. When a path fails, information from the failure can also incorporated into the planner's knowledge of the workspace and lead to the computation of a new maximum expected utility path.

7.1.1 The Uncertainty Roadmap Planner

The following describes the three components the new utility-guided planner: constructing a roadmap, searching for a path with maximal utility and guiding sensing using past motion planning.

Construction

The roadmap constructed by the planner combines several features from previous planning algorithms. Like methods for planning in dynamic environments [45, 65, 109] cycles are introduced in the roadmap to add redundancy. The validation of edges is also delayed until query processing. While other methods [14, 21, 78] do this to improve computational efficiency, the presence of uncertainty introduces additional reasons why this is useful.

Delaying the concrete construction of the roadmap until queried has two significant benefits. First, it eliminates the memory and computational costs of maintaining redundant paths. Edges are only created in the roadmap as required for the solution

of specific queries. Finally, delaying the construction of the roadmap ensures that the most up-to-date information about the workspace is used for construction. The robot is constantly acquiring new information about the workspace, either from sensing or from other planning. Delaying the evaluation of edges until they are required means that the latest knowledge is always used to evaluate their expected utility.

For the selection of nodes in the roadmap, the uncertainty roadmap is sampling-strategy agnostic. Numerous sampling strategies for selecting configurations have been proposed [1, 38, 22, 74] including one that directly incorporates uncertainty into the sampling strategy. Any one of these can be used to select the samples that define an uncertainty roadmap. To effectively distinguish the contributions of the approach from those of any particular sampling strategy, uniform sampling is used in the experimental evaluation (Section 7.3).

Querying

The A* [89] algorithm is used for path queries and to construct the uncertainty roadmap. A* is a general approach for searching implicitly defined graphs. For concrete implementations it requires a cost function, a heuristic function, and a way to obtain a node's children. Note this A* cost is different than the cost of path failure.

The function for estimating the cost of an edge in A* is:

$$G(e) = P(e = \text{obs.})C(e) + \frac{P(e = \text{free})}{U(e)}.$$

$P(e(q_i, q_{i+1}) = \text{obstructed})$ is the probability that the edge is obstructed. This probability is estimated using knowledge of sensor error. The function $C()$ measures the cost of an edge failure. The function $U()$ measures the utility of the edge. Because the A* heuristic function finds paths with minimal cost, $1/U(e)$ is used in this expression. This favors edges with high utility, but ensures that every edge has at least some cost. The experiments use constant cost and utility functions for edges. This imple-

ments a path utility function that favors shorter configuration space paths. Other edge utility functions such as kinematic conditioning could easily be substituted. The A* heuristic function $H(q_i)$ uses the Euclidean distance in the configuration space. This keeps the planner "on target" toward the goal.

During the node expansion step of A*, the implicit uncertainty roadmap connected to a particular configuration is computed. Neighbors of this configuration are selected using the traditional PRM algorithm: The set of nearest neighbors to the configuration are found and returned as possible connections. The planner estimates the probability that the edge leading to each neighbor is obstructed. If this probability is greater than a threshold, the neighboring configuration is ignored. Otherwise, an edge is added to the uncertainty roadmap. Because multiple A* searches may be applied to the same roadmap, the probability associated with each possible edge is cached to avoid redundant computation. This cache is invalidated when relevant further sensing is performed. When A* search finds a path, the path and its certainty are returned to a higher level planner.

Exploration

Until a satisfactory path is found, the planner alternates path planning with guided sensing of the environment. Sensing the environment costs both time and energy. To maximize planner efficiency, only required sensing should be performed. The current best path suggested by the planner provides information about the areas of the workspace that are relevant to the particular path query under consideration. There are two different cases for further exploration: The robot can only explore workspace locations adjacent to its current location. Alternately, the robot may be capable of further exploration of any arbitrary location in its workspace, for example using a so-called "smart-room" camera system. Exploration strategies for each of these two different scenarios are described in the subsequent sections.

Exploration restricted by locality

When a robot's sensing is limited to sensors located on the body of the robot, it can only explore areas that it already knows how to successfully move through. This significantly limits the exploration available to the robot. Further, the robot may have no knowledge of obstacles in workspace regions which it has not moved through. When exploration is warranted, the planner selects the *path fragment* with the highest expected utility. A path fragment is a portion of a path beginning at the robot's current location and terminating somewhere before the goal location. A fragment may fail to reach the goal for one of two reasons: because of uncertainty, the expected cost of subsequent edges in the path is too high or because the roadmap graph is incomplete. In either case, the robot physically executes the path fragment, refining nearby obstacles as it moves. This extends the "frontier" of the robot's knowledge as illustrated in Figure 7.1. Because exploration proceeds along path fragments selected for their utility, exploration is focused on relevant areas. Energy and time are not wasted on obstacles that are not useful for a particular path query. This is shown in Figure 7.1. Knowing the location of the chairs in the lower right of the image is not necessary to reach the goal location. Consequently, those obstacles are not examined by the robot. Additionally, only partial knowledge of the chair on the far left is needed. Despite error in the robot's perception of the obstacle it is still capable of computing a collision free motion.

Unrestricted Exploration

Given appropriate sensor resources, the robotic planner is capable of exploring any arbitrary location in the workspace. In this case, rather than considering the locations which are currently reachable by the robot, the algorithm sorts *all* edges in the selected path by their uncertainty. Edges with the greatest uncertainty are explored first. Highly uncertain edges have high exploration utility for two reasons.

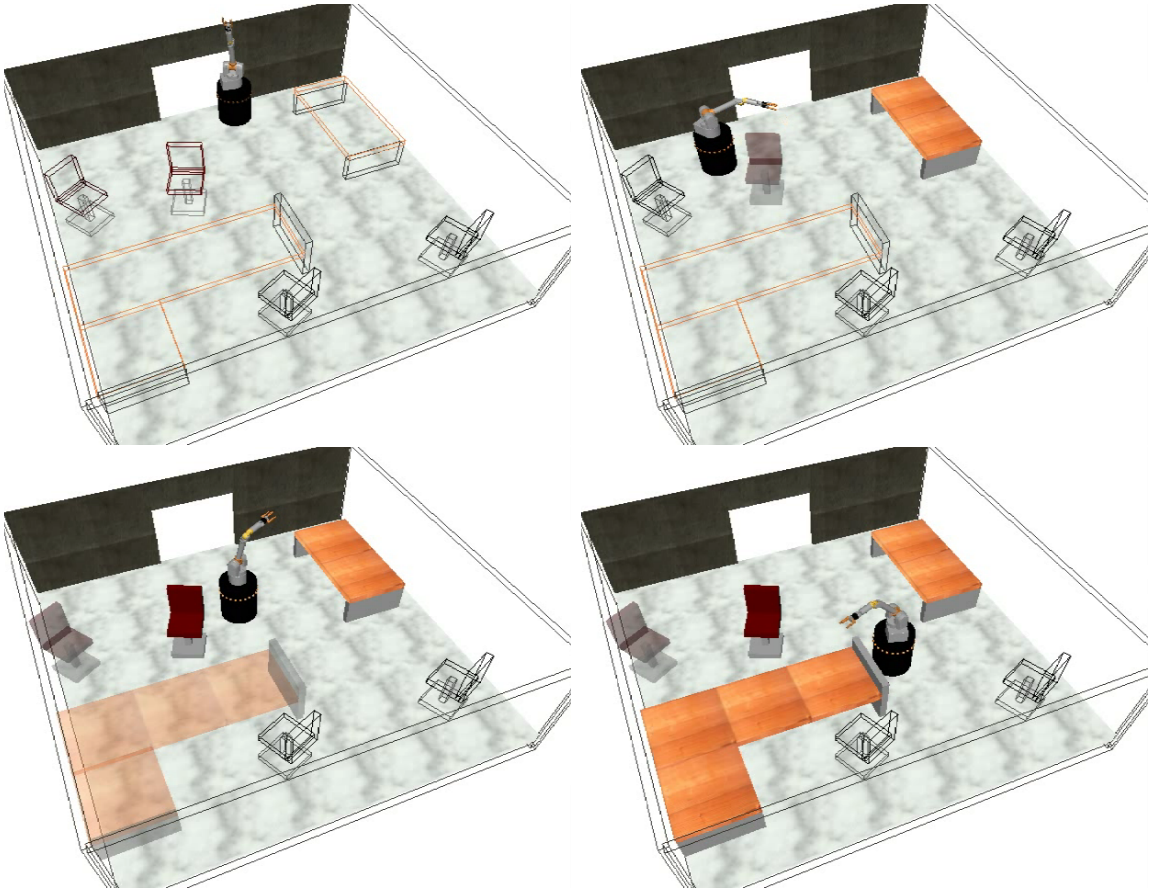


Figure 7.1. The planner in operation, finding a path between the doorway and the desk. Obstacles shown as wireframe are unknown to the planner. Transparent obstacles are partially localized. Opaque obstacles are completely known by the planner.

If they are still believed to be free after sensor refinement, the corresponding reduction in uncertainty maximally increases the expected utility of the path. If they are found to be obstructed, further sensing is not wasted on refining other edges in the path that are more likely to be free. The computational expense of refining every edge in the sorted list is not warranted. Many edges are already highly likely to be collision-free and do not require further refinement. Consequently, the question is: What fraction of the edges should be refined? In the implementation of the utility-guided planner, this fraction was left as a runtime parameter and I performed experiments to observe the effect of different fractional values. Regardless of the fraction the planner receives new workspace information from additional sensing and performs a new A* search to identify the path with the current greatest expected utility. In areas of the workspace that haven't been refined, previous uncertainty values are cached to render re-planning more efficient.

7.2 Modeling sensor error

Calculating the expected utility of a path requires estimating the probability that an edge has been erroneously labeled as collision-free. To estimate this probability, it is important to understand the possible causes of such an error. These causes are entirely dependent upon the method of integrating sensor data to represent the workspace. The following considers two different approaches to representing the workspace: first an occupancy grid representation constructed from range sensors such as lasers or sonars and second, estimating the position and orientation of known objects through localization in a digital camera system.

7.2.1 Range-based Workspace Representation

An occupancy grid subdivides the workspace into series of cells. Although some occupancy grids allow mixed or probabilistic labels for cells, this work assumes that

the grid is binary; cells are marked as either obstructed or free. To evaluate the state of a configuration, the workspace pose of the robot in the configuration is computed. If any of the cells that overlap this location are obstructed, the configuration is considered obstructed. If all of the cells are free, the configuration is free.

Occupancy grids are typically constructed using information from a sensor such as a laser or sonar that returns a distance from the sensor to an object in the environment. The noise and error in the distance returned by a sensor is specific to each sensor. Nonetheless, a general understanding of the resulting errors in the occupancy grid can be developed. Regardless of its specifics, a range sensor casts a ray out from a central point until it strikes an obstacle in the environment. The length of the ray is used to build the occupancy grid. The cells intersecting with the portion of the ray that extends beyond the measured distance are marked as obstructed. When sensor error occurs it results in the mis-labeling of cells. Most commonly this occurs because of noise in the range data. If the distance is slightly too long, a cell that is obstructed will be labeled free. If it is too short, a cell that is free will be labeled obstructed. Mis-labeling may also occur because of occlusion when the "shadow" of the ray cast by the range-finder is also thought to be obstructed. The following sections describe two approaches to modeling this erroneous mislabeling. These models predict the underlying true state of roadmap edges the workspace. Both models assume that they receive as input a series of observations of the robot's state (obstructed or free) coarsely sampled along the length of an edge. From these observations the models calculate the expected true state of the edge (obstructed or free) and the probability of this state.

Naive Bayes

A naive Bayes model [89] derives its name from its assumption that all observations are independent. For edge checking, this assumption is false, hence the model's

naivete. However, in many cases the failure of this assumption does not significantly impact the model's predictive performance and it greatly simplifies calculations. By assuming each observation of the state of the robot is independent, the probability of a hidden state given a series of observations is simply the product of each individual probability. For this work, the probability of an obstructed edge is given by:

$$\prod_{j=1}^{n} \frac{P(q_j = x | e = \text{obs.}) P(e = \text{obs.})}{P(q_j = x | e = \text{obs.}) P(e = \text{obs.}) + P(q_j = x | e = \text{free}) P(e = \text{free})}.$$

Applying this model, requires parameter estimates: $P(\text{obs.})$, the probability that an edge is obstructed, $P(q_j = \text{obs.}|\text{obs.})$ the probability of an obstructed observation given an obstructed edge and $P(q_j = \text{obs.}|\text{free})$ the probability of an obstructed observation given a free edge. Fortunately, estimates of these parameters seem to be generally applicable across similarly structured environments, such as all office spaces. This means that the parameters for the model can be estimated from complete observation of a a single known workspace and subsequently applied to numerous other similar environments.

Hidden Markov Models

Hidden Markov models (HMMs) are a graphical model [89]. Unlike naive Bayes models, HMMs do not assume that each observation is independent. Thus they more accurately represent sequential observations like the examination of an edge connecting configurations. The noisy examination of a trajectory is modeled as the simple HMM shown in Figure 7.2. There is a vertical slice in the model for each sequential observation of a configuration along the linear edge. In each slice the hidden node represents the true, unknowable, state of the configuration. The observed node contains the noisy observation generated by the planner's representation of the workspace.

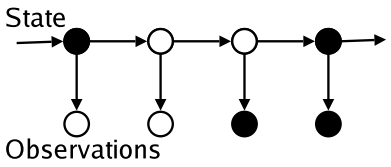


Figure 7.2. The hidden Markov model used for predicting the state of an edge given a set of observations.

Estimating the obstructed probability of a particular sequence of observations begins by labeling the observation nodes with the observed values. The Viterbi algorithm [89] is used to calculate the maximum likelihood sequence of hidden states given the observations. This estimate of the hidden state is used to predict if the edge is obstructed or free. The likelihood of the sequence of hidden states is used to estimate the certainty of this prediction.

7.2.2 Localization Workspace Representation

An alternative method for representing the workspace is to assume that the geometry of all objects in the workspace is known a priori. This assumption is fairly safe in lightly controlled environments such as an office or worksite, where the set of objects present is fairly constant and persistent. Thus, the time spent modeling all objects is worthwhile and the likelihood of encountering an object that is not known is small.

When objects are known, they can be identified and their pose estimated using stereo vision or other sensory input. This process produces what I call a *localized obstacle* representation of the workspace that differs from an occupancy grid. In the localized obstacle representation, each object is represented individually and collision checking is performed using the known object models placed in the pose determined by sensing. In such a representation, error occurs in the mis-identification of an object's pose. By assuming this error follows a known distribution, the certainty of

each configuration space observation can be calculated. This work assumes that the error follows a Gaussian distribution.

Gaussian Error Models

Whether a configuration is obstructed or free, the certainty of that observation is related to the magnitude of localization error that would invalidate the observation. The following only examines translational error, but the same approach could be extended to rotational error. This model also assumes that error is equally likely in all three translational axis.

When a configuration is tested to determine if it is colliding or free, the geometric algorithm for collision detection also returns the penetration depth of the collision, if the configuration is obstructed, or the distance to the nearest obstacle, if the configuration is free. This distance can be used to determine the certainty that the calculated state of the configuration (obstructed or free) is in fact the true state of the configuration.

For the calculated state of the configuration to be incorrect it is necessary that the error in the localization of an obstacle must be greater than either the penetration distance or the distance to the nearest obstacle for an obstructed or free configurations respectively. Assuming that the error in the localization of an obstacle follows a known distribution allows calculation of the probability that the error in localization has sufficient magnitude to invalidate the state of the configuration. The following expressions assume a Gaussian distribution of localization error, but they could easily be modified to incorporate some other distribution whose form is known. The probability that the magnitude of error that lies within a range that doesn't invalidate the observation is given by the cumulative distribution function (CDF) of the distribution, in this case the Gaussian:

$$\frac{1}{2}(1 + \text{erf}(\frac{d}{\sigma\sqrt{2}})).$$

Thus the probability that the observation of the configuration's state is incorrect is:

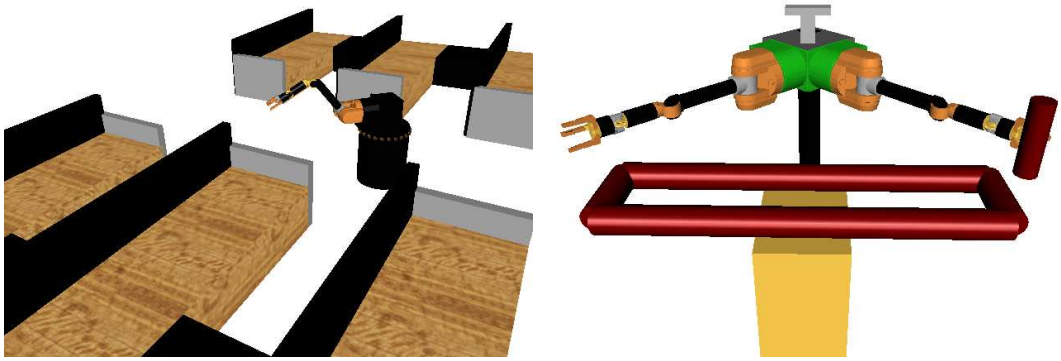
$$1 - \frac{1}{2}(1 + \text{erf}(\frac{d}{\sigma\sqrt{2}})).$$

In these expressions, "erf" is the Gauss error function and d is the penetration depth or distance to nearest obstacle. It is important to note that this only approximates the true probability of error. It only considers interactions between the robot and the localization of the closest obstacle. In situations with significant error, it is possible that localization of other obstacles will also have an effect. In practice, this approximation seems to work reasonably well. Because collision distances are challenging to calculate, better approximations are possible but incur significantly greater computational costs.

7.3 Experiments

Sampling-based motion planners are generally evaluated by comparing execution times for difficult planning problems. This method of evaluation is not appropriate here. Instead, I compare how well the proposed sampling-based planner can handle uncertainty in comparison with the traditional PRM method [52]. Since I am not concerned with execution time, this is an adequate comparison. Using planners that employ non-uniform sampling would make an experimental evaluation more difficult, due to interactions between uncertainty and the sampling distribution.

Several experiments are necessary to demonstrate the suitability of the approach described in previous sections. This evaluation addresses several important questions. First, can the models discussed in Section 5.4 provide reasonable values for



(a) 10-DOF Mobile Manipulator (b) 14-DOF Humanoid Torso

Figure 7.3. The experimental workspaces and robots used to evaluate the predictive roadmap.

use in constructing the predictive roadmap? Second, does the predictive roadmap significantly improve the ability of a motion planner to compute collision-free paths in the presence of uncertainty? Finally, can certainty information be used to guide further sensing to refine the planner's perception of the environment.

To answer these questions, the planner was run in three simulated worlds designed to resemble real-world environments. Two of these are pictured in Figure 7.3: a 10-DOF mobile manipulator platform in an office environment and a 14-DOF humanoid torso in a construction environment. A third world with a 2-DOF circular robot in the same workspace as the mobile manipulator was also used. Experiments were run in each of these worlds simulating both methods of representing the workspace. Error in the occupancy grid representation was simulated using an adjustable probability p_{err} that any particular cell in the grid is mis-labeled. To simulate error in the localized obstacle representation, translational error was added to each obstacle's true position. The magnitude of the error was sampled from a Gaussian distribution and differed for each obstacle.

7.3.1 Modeling

I first examine the models used to assign values to the predictive roadmap.

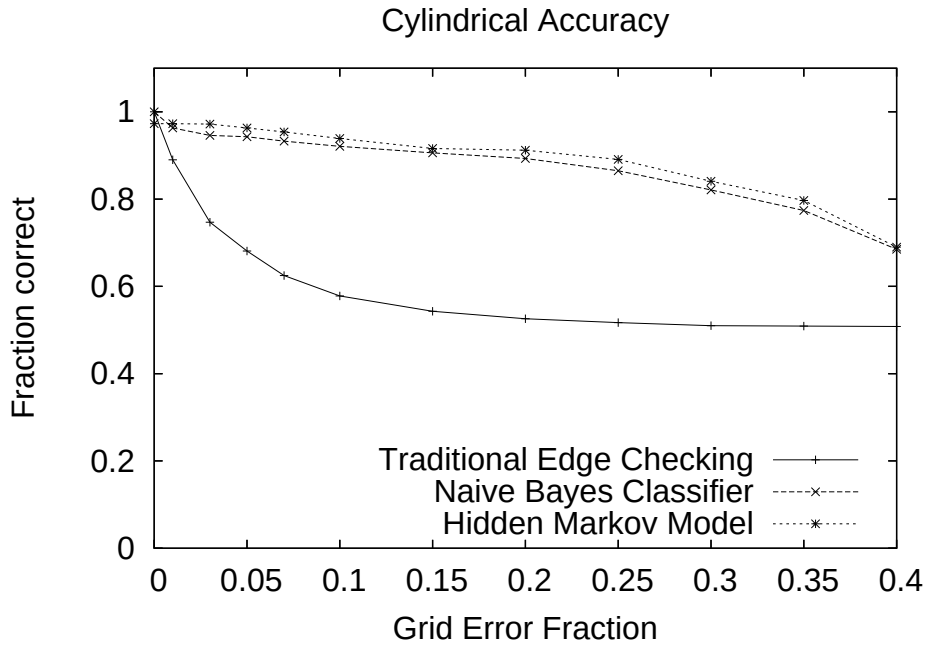
Occupancy Grid

The evaluation of the naive Bayes and hidden Markov models used 500 free and 500 obstructed edges in configuration space. These edges were chosen uniformly at random from the set of edges that were shorter than the radius used for the nearest neighbor query in motion planning. They were intended to represent typical edges encountered in motion planning. For each edge, each model was asked to predict if the edge was obstructed or free. The traditional edge checking model simply returns obstructed if any configuration along the edge is observed to be obstructed. The fraction of the edges that each model correctly predicted as a function of the probability of error is shown in Figure 7.4.

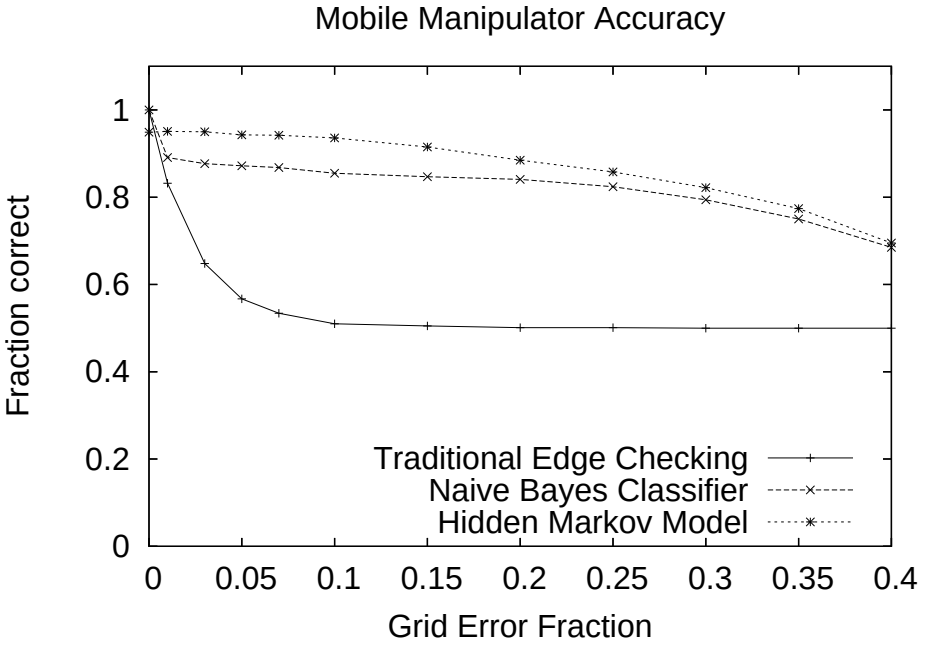
In all three worlds, as error increases traditional edge checking quickly devolves into predicting that all edges are obstructed. This is because the probability of moving along an edge and not receiving an erroneous observation rapidly becomes quite small. This means that even completely free edges are likely to result in an obstructed observation. In contrast, both the naive Bayes and hidden Markov models are significantly more robust to error. Interestingly, HMMs only slightly outperform naive Bayes models in the environments. This indicates that the added accuracy does not justify the increased computational cost of the HMM. Consequently, in the planning experiments the naive Bayes model for predictions was used.

Localization

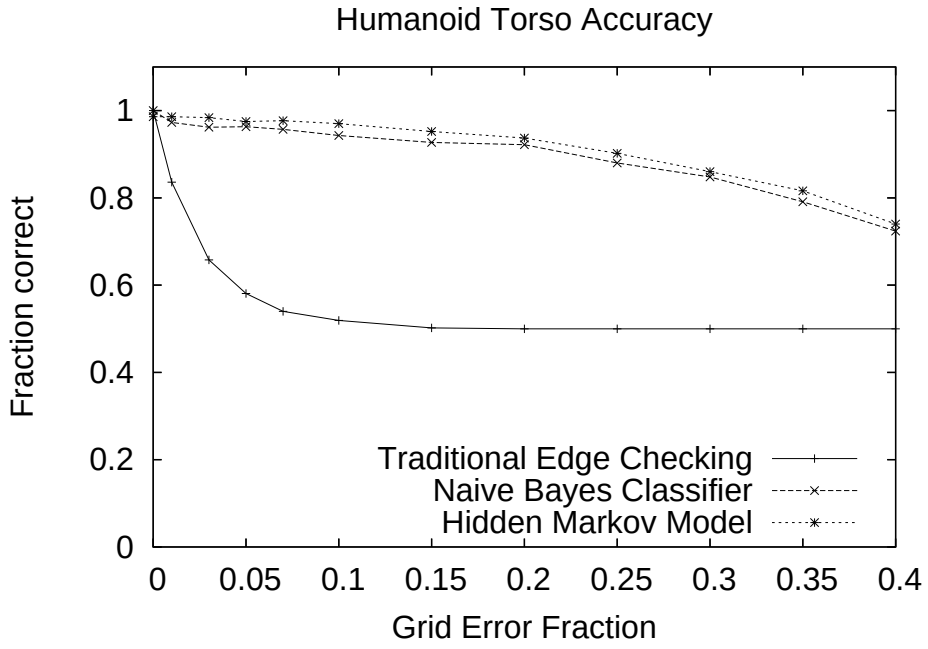
Unlike the occupancy grid model, the model of uncertainty in localization can not be used to provide predictions about the state of an edge, instead it provides the reliability of the workspace model's observations. This uncertainty is used to bias the posture of the robot toward configurations that are more certain to be free. An image



(a) Two Dimensional Circular Robot



(b) 10-DOF Mobile Manipulator



(c) 14-DOF Humanoid Torso

Figure 7.4. Edge predictions accuracy for HMM, Naive Bayes and traditional edge checking as a function of error for three different robots with varying degrees of freedom.

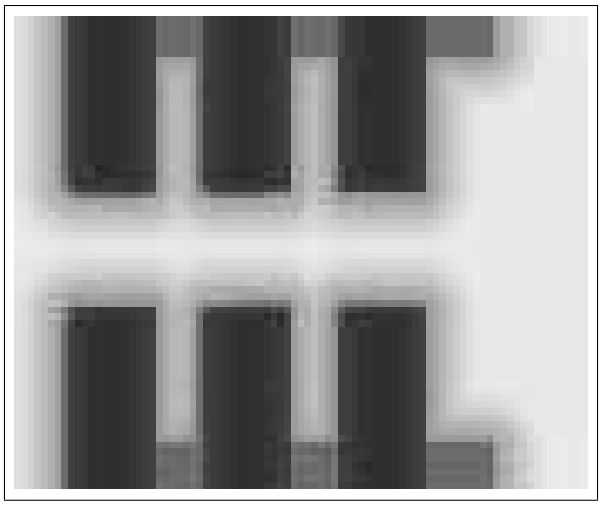


Figure 7.5. A graphical representations of the uncertainty function for the 2-DOF circular robot using a localized obstacle model of the workspace. Black indicates absolute certainty of obstruction, white indicates absolute certainty of free space.

of this uncertainty function for the 2-DOF circular robot in the office cubicle environment is shown in Figure 7.5. In the image, black indicates obstructed with absolute certainty and white indicates free with absolute certainty. This image indicates that a planner which selects motions biased toward minimizing uncertainty function will choose configuration-space paths that are maximally distant from obstacles and thus more likely to be unobstructed.

7.3.2 Planning

In addition to assessing the uncertainty models, experiments were used to evaluate the use of these models in predictive roadmap motion planning. To evaluate the planner I generated a series of random path queries for each environment. The traditional implementation of PRM planning and the predictive roadmap planner were asked to find a path that satisfied the query. The amount of error present in the workspace representation varied. The experiments measured the percentage of queries that the planner could solve correctly and the average runtime that the planner took in computing these solutions. All of the path queries had solutions given accurate rep-

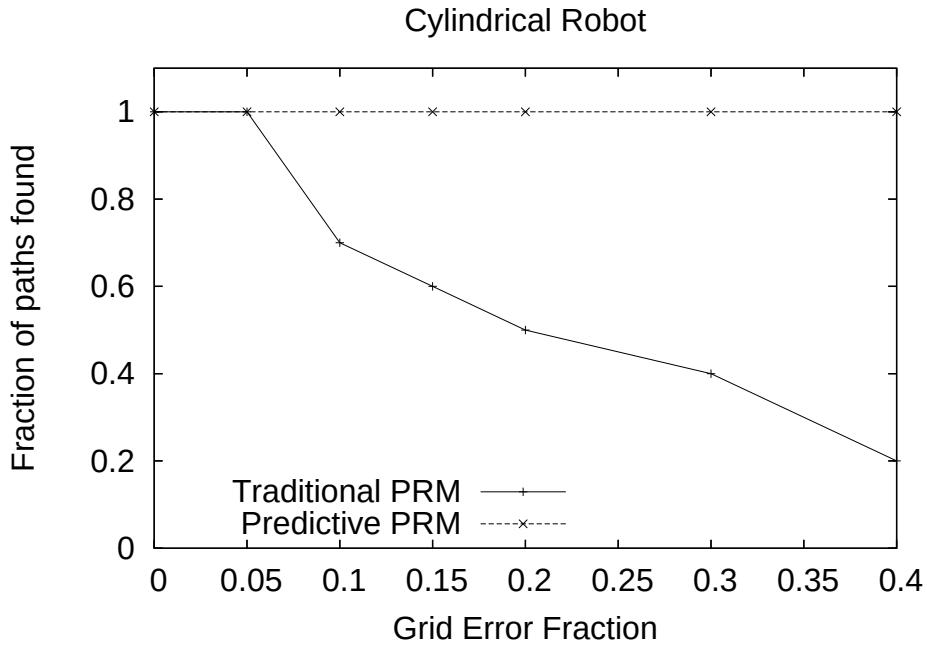
resentations of the configuration space. All experiments were implemented in C++ and were run on a 3Ghz Pentium 4 with 1 GB of RAM running the Linux operating system.

Occupancy Grid

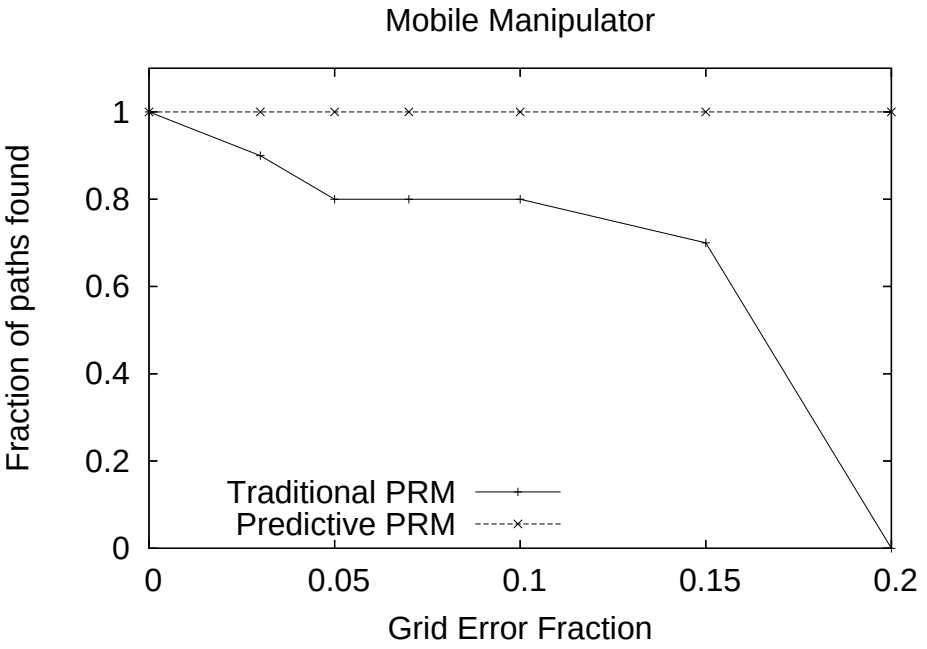
The experiments using an occupancy grid representation introduced error by increasing the probability that a cell in the grid was mis-labeled. The naive Bayes model was used by the predictive planner to estimate the state of edges in the predictive roadmap. The fraction of path found and the average runtime for each planner in each environment as a function of error is shown in Figures 7.6 and 7.7.

These graphs indicate that because the predictive roadmap algorithm is aware of uncertainty and the possibility of error it is significantly more capable at finding a path as error increases. For the traditional PRM algorithm there were two main causes of failure. First, error in the workspace representation caused the initial or goal states to be mis-classified as obstructed. Second, I halted the operation of the traditional PRM algorithm at 30 minutes, path queries that were halted, were considered to have failed.

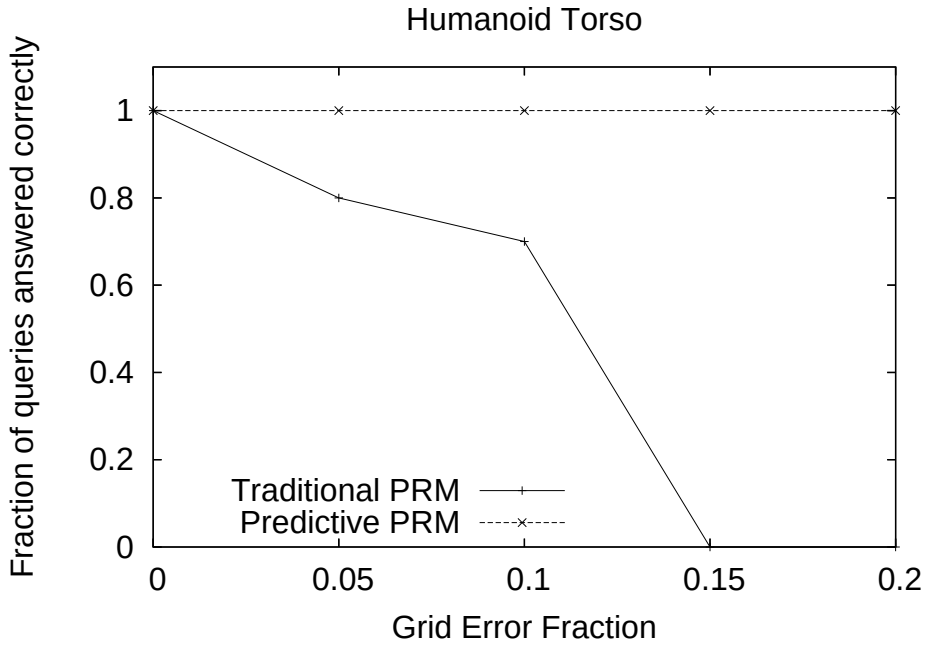
Interestingly, when there is no error, the excess computation associated with the predictive roadmap makes its average runtime slower than traditional PRM planning. As error increases, the runtime of traditional PRM planning increases significantly faster than predictive roadmap planning and even when the probability of error is only 5%, predictive roadmap planning is more efficient than traditional PRM planning. In large part this is because error has a similar effect to narrow passages on PRM planning. As error increases the probability of finding a collision free edge significantly decreases, requiring traditional PRM planning to perform significantly more exploration.



(a) Two Dimensional Circular Robot



(b) 10-DOF Mobile Manipulator



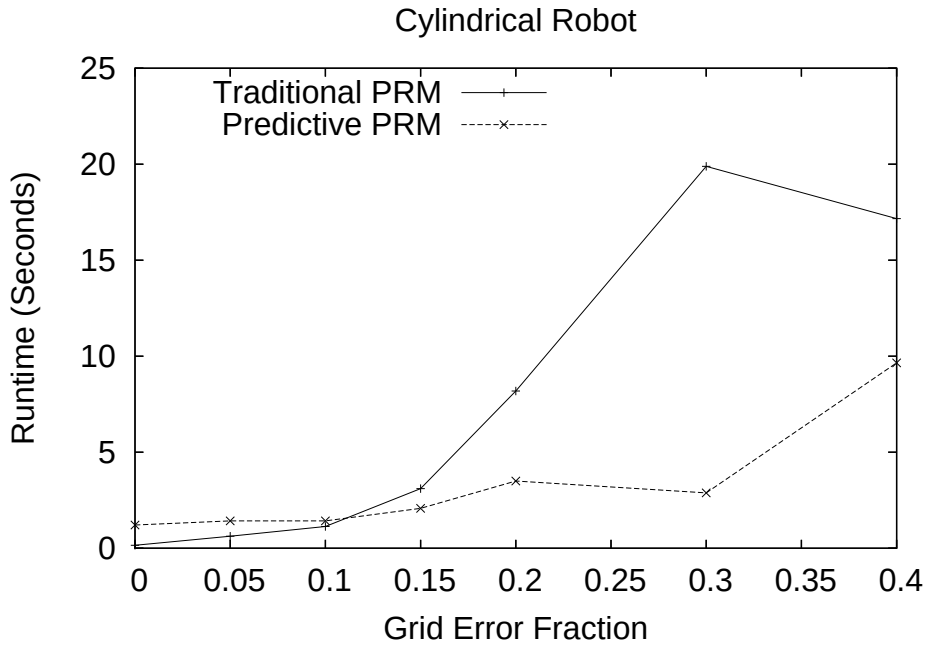
(c) 14-DOF Humanoid Torso

Figure 7.6. Fraction of paths successfully found by tradition PRM and the predictive roadmap algorithm as a function of occupancy grid error.

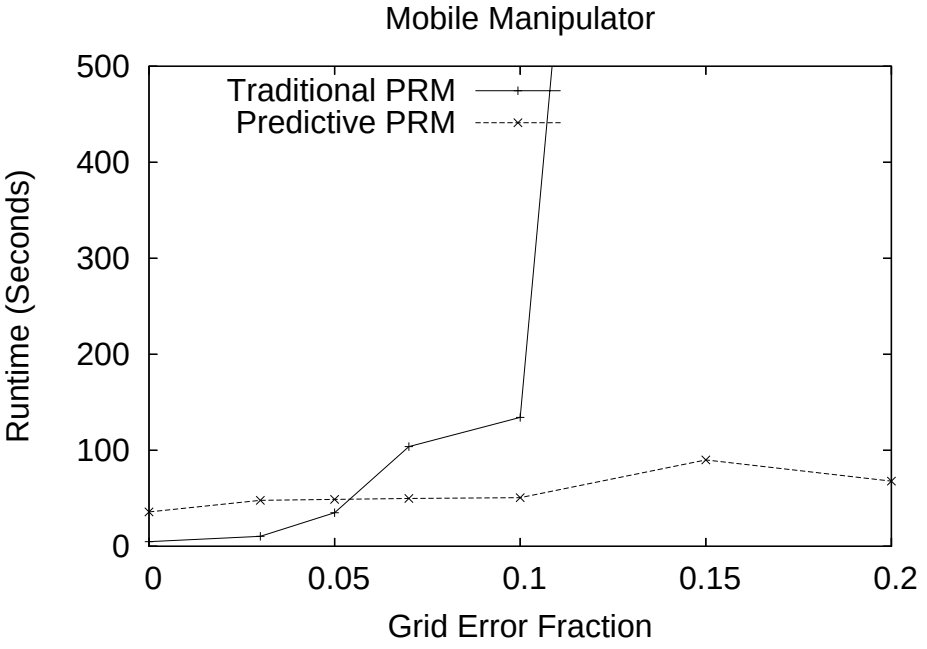
Localization

Predictive roadmap planning was also used with a localized obstacle model of the workspace. As before error was introduced into the workspace representation. This time by adding Gaussian noise to the true location of each obstacle. Each planner was asked to find a path between a random pair of configurations drawn from opposite sides of the workspace. Once the planner had solved they query, the correctness of the path was tested against the true state of the workspace. Whether or not the path computed by the planner was actually collision free was recorded. Experiments were run comparing predictive PRM with traditional PRM.

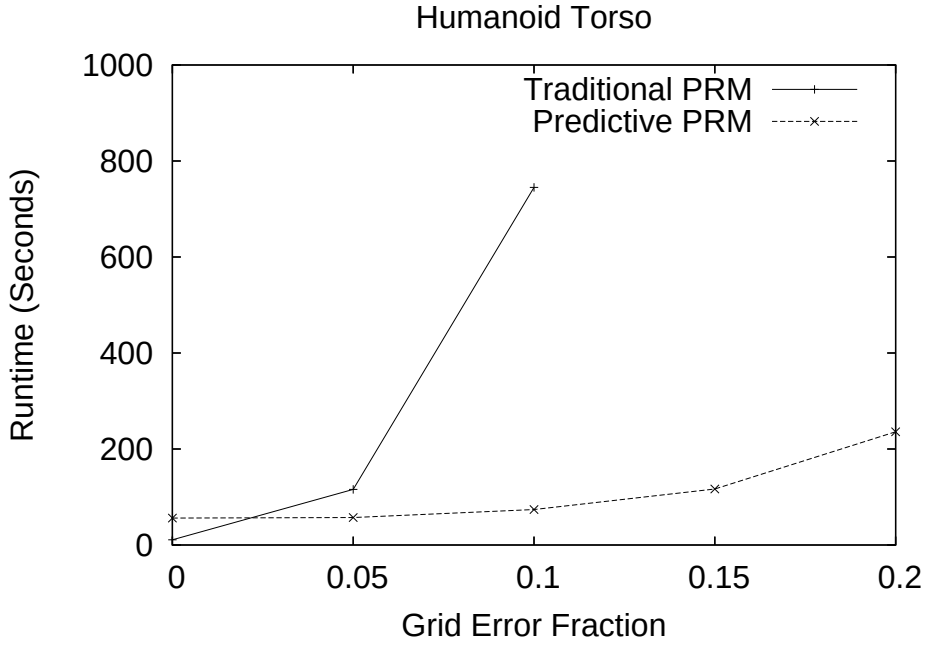
The experiments were also used to test unrestricted utility-guided exploration. Once a path was found by predictive PRM, different refinement thresholds were used to ensure the paths correctness. The refinement percentages indicate the percentage of edges in a path that will be refined. When refining a path, predictive PRM sorted



(a) Two Dimensional Circular Robot



(b) 10-DOF Mobile Manipulator



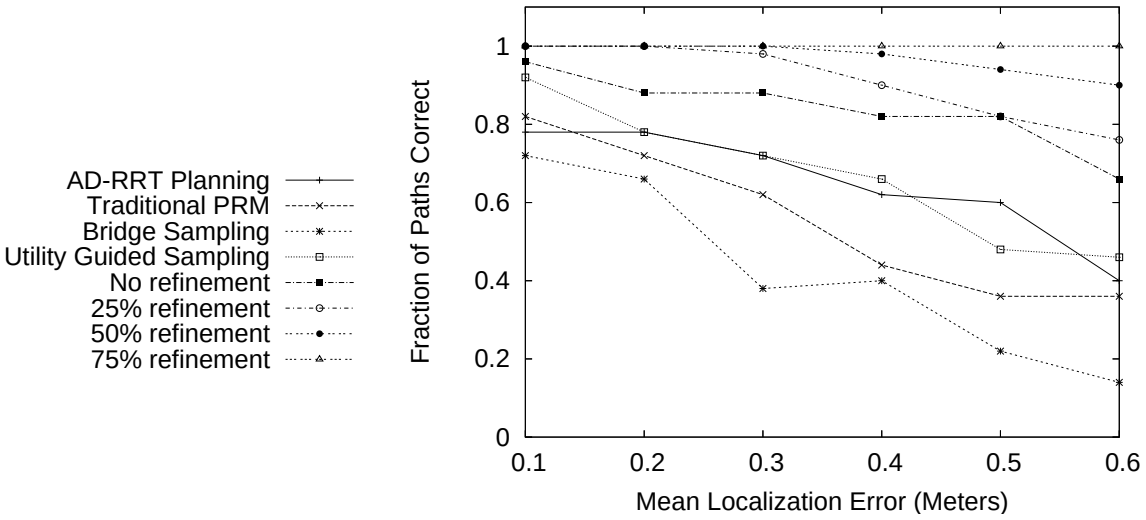
(c) 14-DOF Humanoid Torso

Figure 7.7. Runtime required to successfully find a path as a function of occupancy grid error.

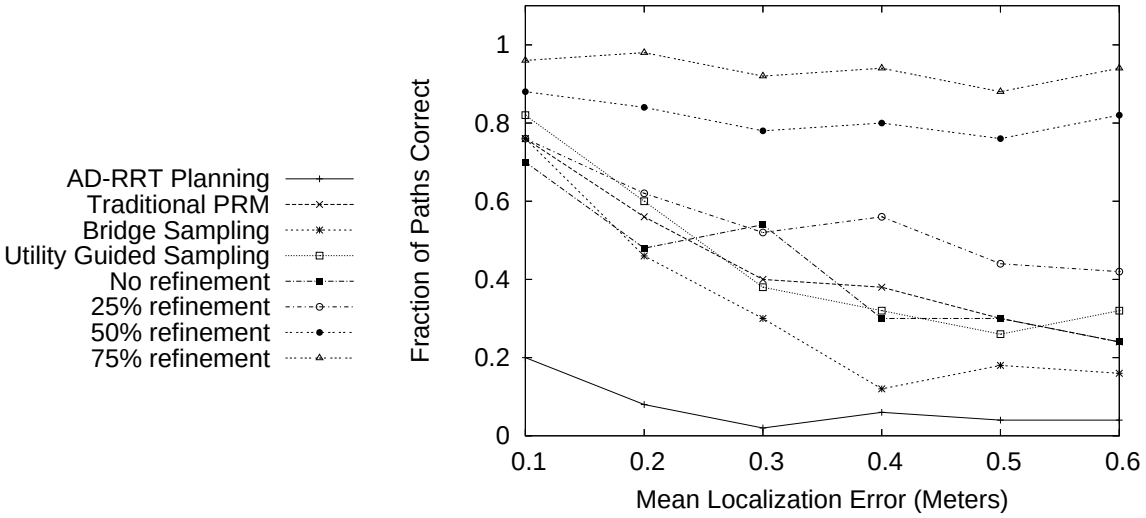
all of the edges in a candidate path by their probability of being free, with the least probable edge refined first. In these simulation experiments, the refinement step removed the localization error for obstacles near to the edge and then re-checked the edge to ensure that it was collision free.

Figure 7.8 shows the fraction of paths returned by each planner that were actually collision-free as a function of mean localization error in meters.

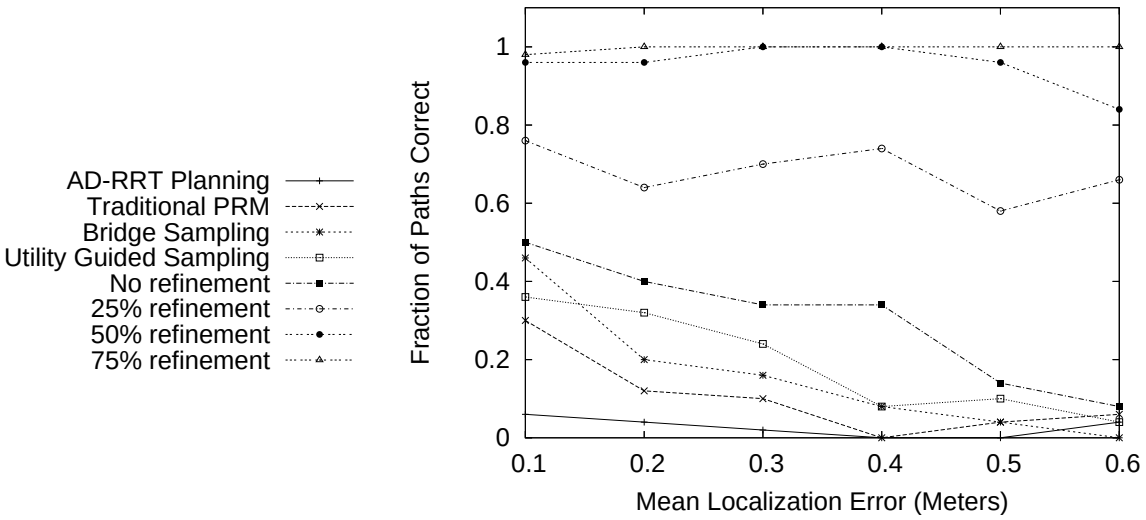
In two out of the three worlds, predictive PRM without any refinement outperforms traditional PRM, selecting paths that are more likely to be unobstructed. In the mobile manipulator in the office environment, traditional PRM outperforms predictive PRM with no refinement, though their performance converges as error increases. This demonstrates that in the absence of refinement, predictive PRM is only as reliable as its measure of certainty. The certainty metric for the mobile manipulator is not as accurate since many different parts of the robot contribute to a possible colli-



(a) Two Dimensional Circular Robot



(b) 10-DOF Mobile Manipulator



(c) 14-DOF Humanoid Torso

Figure 7.8. Fraction of correct paths found by each planning algorithm as a function of localization error

sion, examining only the closest point to an obstacle produces inaccurate information. Fortunately, even with a little refinement, predictive PRM significantly outperforms traditional PRM planning.

In all of the worlds, predictive PRM without refinement outperforms AD-RRT planning. Not shown in the graphs are results for traditional RRT planning which is even worse. Random tree planning performs significantly worse that other planners in the presence of uncertainty because of details in the RRT algorithm. RRT expands all branches of the tree until they reach an obstruction, consequently many nodes in the tree are quite close to configuraiton space obstacles. When the actual location of these obstacles is uncertain, many of the paths through the tree are actually invalid. This is further demonstration that reasoning about uncertainty is necessary for successful real-world planning.

The graphs also show that, as expected, more refinement improves the reliability of the paths returned by the predictive planner. Interestingly, even relatively little refinement showed significant improvement in plan reliability and 50% refinement

to an average of over 90% reliability. This shows that guided exploration is able to target sensor refinement to relevant regions of the workspace while relying on a coarse representation where it is appropriate. Sensing is expensive both in terms of time and computation. Reducing the required sensory refinement by 50% leads to significant increases in planner performance.

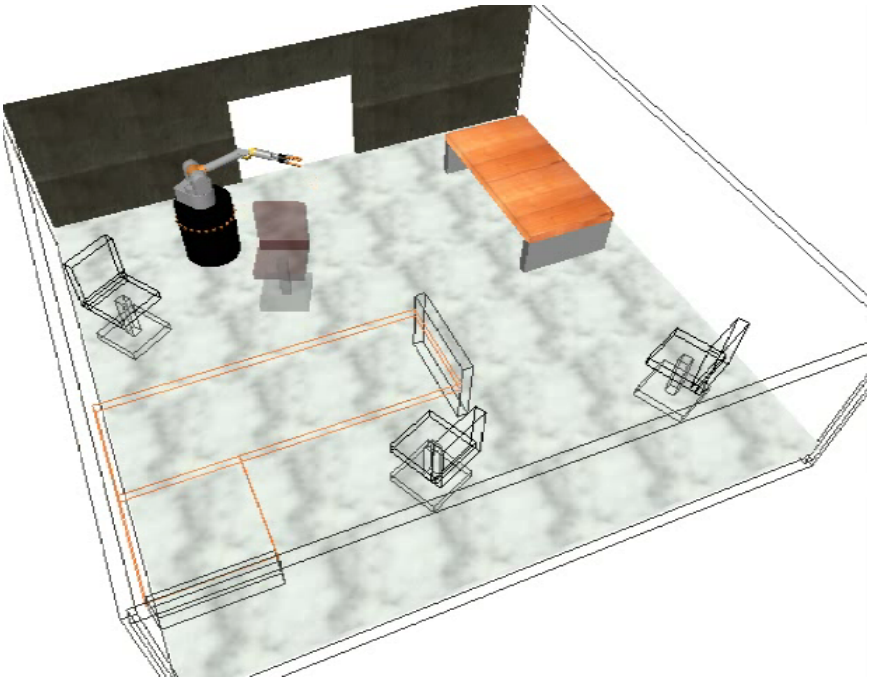
Unlike planning with the occupancy grid representation, the runtime of the planner did not vary significantly as error increased. For the two more complex workspaces the predictive roadmap planner is nearly an order of magnitude slower than traditional PRM planning. This is largely due to the repeated nearest obstacle and distance queries incurred in evaluating the certainty of each edge.

Restricted Exploration

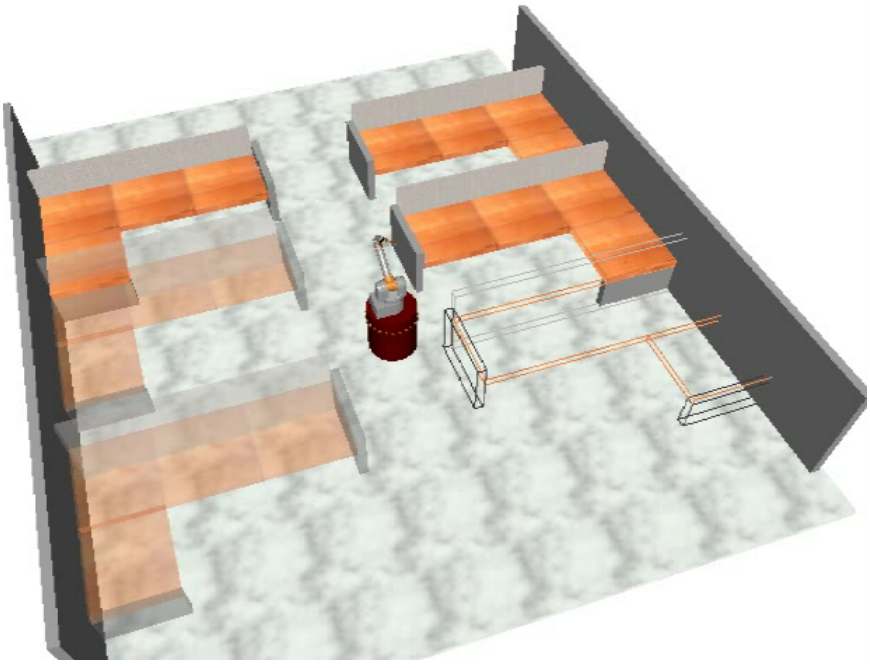
I also performed experiments where the planner's exploration was restricted to a region adjacent to the robot's current location. In each experiment the robot was asked to go from a position on one side of the room to a position on the other side of the room. In the cubicles world, this meant going from one side of the cubes to the other, in the office world, this meant moving from the doorway around to the side of the desk. The worlds are shown in Figure 7.9.

Initially, the robot had no knowledge of any obstacles except the ones immediately adjacent to its starting position (one cubicle in the cubicle environment and the doorway in the office environment).

When an obstacle was discovered it was localized by the robot ("Obstacles Localized" in the graphs). This localization Gaussian distributed error with a mean of 0.4 meters in each dimension. The robot then had the option of subsequently refining the localization to eliminate error ("Obstacles Refined"). The planner ran until it computed a successful collision free plan.

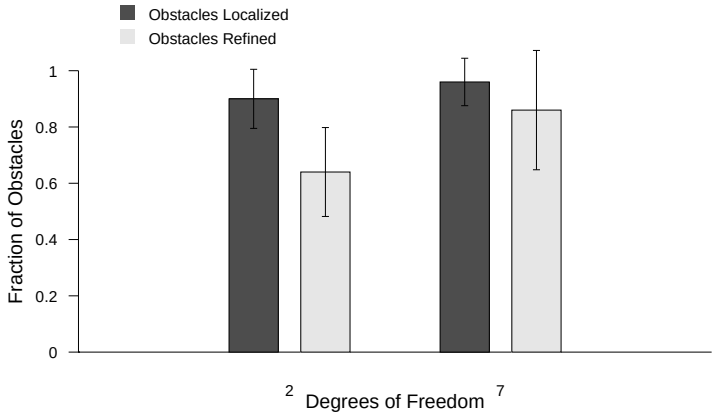


(a) Desk world

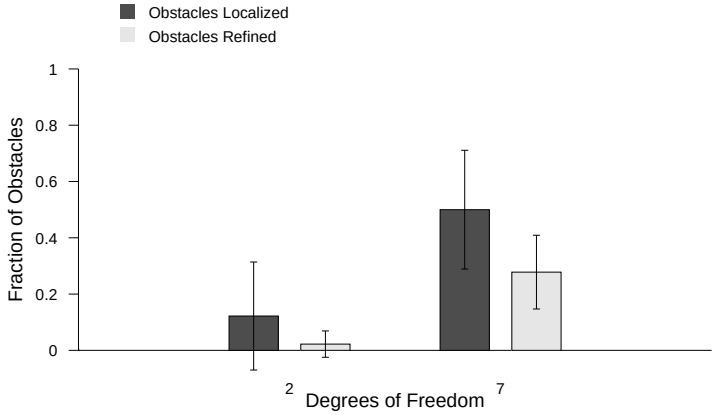


(b) Cubicles world

Figure 7.9. The two experimental worlds for restricted exploration experiments



(a) Results for the cubicles workspace



(b) Results for the office workspace

Figure 7.10. Experimental results for uncertainty planning with restricted exploration

For the two worlds I ran two experiments, one planning just for the base (2-DOF) and one planning for the base and the arm minus the wrist (7-DOF), this results in four scenarios.

For each of these four scenarios, ten experiments with different random seeds were run. The number of localizations and refinements were recorded for each run. Figure 7.10 shows the results of these experiments. The graphs show the average and standard deviation of the number of explorations for each DOF in each world.

There are several interesting things to note in these results. The robot always makes fewer refinements than localizations. This is an obvious result, but it shows that notion of sensing at different resolutions to improve planner efficiency is a good one. Further, the robot's never completely explore the workspace. This also shows that guided exploration is more efficient.

The cubicles workspace requires the exploration of a significantly greater fraction of the workspace obstacles. This is because all of the obstacles in the environment are at least partially relevant to a successful motion plan. On the other hand, in the desk environment, many of the obstacles (for example the two chairs in the lower right) are not relevant to the requested motion plan, consequently they are never examined by the planner. This results shows that the planner can differentiate between relevant and irrelevant obstacles and focus its sensing accordingly.

Another interesting result is that increased degrees of freedom in the same experimental world results in more exploration. This was not an expected result. I attribute it to two factors: First, the uncertainty metric is not as accurate in higher degree-of-freedom worlds because the error in localization is taking place in the workspace, while the uncertainty is being evaluated in jointspace. When the joint-space and workspace match up nicely (as in 2d) it works better. Second, the configuration space is larger and the passages are narrower. I think this later fact is particularly interesting because it implies another reason for minimizing the degrees-of-freedom you plan.

Chapter 8
Conclusions

Successful deployment of autonomous robots into the real world requires that these agents be capable of physically interacting with the world with sophistication that is on par with their human counterparts. One important skill required for these interactions is *motion planning*. Motion planning is the task of designing movements for complex limbs through obstructed environments that avoid collisions with either the robot or other obstacles. While there have been significant improvements in the performance of motion planning algorithms over the past decade, there are a number of significant challenges that must be solved to enable the real-world use of motion planning. These challenges include further improvements in planner performance, motion planning in dynamic environments, tasks-specific planning, the integration of planning and control and much more.

This thesis presented a utility-guided approach to motion planning. Utility-guided planning is a general framework for guided robotic motion planning using inaccurate and/or incomplete information. Utility-guided planning relies on two sources of information to guide the planner's exploration: an approximate model using the information currently available to the planner and a utility-function which estimates the value of information gained via exploration. The planner uses the approximate model to predict the expected utility of available explorations and selects the exploration that is maximally useful. This exploration is enacted and information obtained via exploration refines the planner's approximate model. This process of exploration and improvement iterates repeatedly. When sufficient information for solving the mo-

tion planning problem has been acquired, the utility-guided algorithm returns this solution. This approach to planning has several important strengths. First, it is constantly adapting. As new information is obtained it is incorporated into the planner to guide subsequent exploration. Second, each exploration it selects is expected to maximally improve the planner's current state. This constant improvement makes the performance of utility-guided planning approximate an anytime algorithm. Whenever planning is halted, the current solution approximates the optimal solution available given the time allotted for planning.

The utility-guided framework is general purpose and has been applied to two important challenges necessary for motion planning to be feasible in the real world. Utility-guided planning has been used to create new motion planning algorithms for both single and multi-query motion planning. These planners explore by sampling the configuration space. The utility-guided planners select samples with the greatest expected utility to the planner. This improves the runtime performance and overall reliability of the planner, both important considerations for real-world planning. Utility-guided planning has also been used to develop a new planning algorithm that explicitly incorporates uncertainty about the robot's perception of the world into the planning process. When a robot uses sensors to construct a representation of its environment, the resulting representation is necessarily noisy and possibly erroneous. By understanding this inherent uncertainty, the utility-guided planner can select paths which minimize the expected cost of failure. This improves the reliability and success of a robot planning in the real world. It also enables the planner to select motions that are best suited to the specifics of the environment, robot and task.

This thesis has presented a general framework and several concrete algorithms which advance the state of the art in motion planning. The process of developing these algorithms has also afforded a comprehensive perspective on the broader requirements for an effective real-world robotic motion planner. The primary observation

provided by this experience is the necessity of feedback in real-world physical interactions. Successful interaction is defined by a reliance on sensory feedback. Feedback enables a planner to handle two of the significant problems facing any real world robot: dynamic worlds which are constantly changing, and uncertain perception of these dynamic worlds. However, incorporating feedback into the planning process is tricky. Because of their complexity, planners can take several seconds to operate. Consequently, the world can change dramatically while a plan is computed. One approach to solving this problem, is to improve the efficiency of the planner. Creating motion planning algorithms with the kilohertz performance of traditional control methods is a worthwhile, but extremely challenging, task. In the absence of such methods, it is possible to create planners which are efficient enough to operate in slowly evolving dynamic environments. The benefits of increased efficiency are the primary focus of the multi-query and single-query utility guided planners.

However, though the world changes, these changes rarely invalidate all previous information available about the world. The true incorporation of feedback into motion planning involves using past information which is still valid to guide the planner while incorporating sensory feedback to adapt the planner to new information as it is observed. The utility-guided approach to planning under uncertainty is a first step toward such a planner. When the planner is unsuccessful, it uses past information from planning to guide the sensing required to obtain the information need for computing a successful plan. As new sensory information is obtained, previous knowledge invalidated by current sensing is discarded.

In addition to the incorporation of feedback, another important concern of real world planning is uncertainty perception of the robot's environment. Whenever a robot uses physical sensors to identify features in the real world, the process of sensing necessarily introduces errors into the robot's perception. Although the incorporation of sensing and feedback into motion planning enable planners to partially handle

uncertainty. It is also important to directly incorporate uncertainty into the planning process. Feedback enables a planner to react to erroneous perception when it attempts an flawed action which depends on the error. However, incorporating uncertainty enables the planner to identify plans which minimize the possibility of such an error occurring in the first place. In addition to explicitly incorporating feedback, utility-guided exploration also incorporates uncertainty into the planning process. The plans selected by the planner are those with maximal utility. These paths minimize the expected cost (negative utility) of failed execution of portions of the path.

Although I believe that this thesis begins to take steps towards real-world motion planning for jointed robotics. It is clear to me that it does not provide a solution for these problems. Instead, I hope that it opens a door, provides a perspective and a direction for future work. To me, it seems that the central challenge is incorporating many different algorithms running at different speeds serving many different masters. Even with feedback, planners will never run as fast as the PID controllers which keep motions stable, nor will they even achieve the speed of trajectory generation which supplies reference positions to the lower level controllers. Instead, all of these algorithms will need to run at their different speeds, examining the world with the different perspectives that their different focuses provide. Integrating these different processes requires understanding the flow of information between the processes. Past and present work has focused a great deal on how the information can flow down through the hierarchy: planners provide way points to trajectory generators, which in turn provide a reference signal to a control algorithm, but relatively little has been done with regards to bringing sensory feedback back up from the lowest level.

Beyond the integration of planning and simple control, lies the integration of development of multi-objective planning and multi-objective control. Both of these topics have been considered individually, however, their integration has not. It is clear that true exploitation of the possibilities afforded by the world requires simultaneously

achieving multiple objectives. One need only watch a person carrying their lunch in their hands open a door with their foot to know that this is true. Multi-objective behavior is another step toward effective robotic in real-world environments.

Making motion planning algorithms capable of planning movements for robots in the uncontrolled real-world environments will significantly advance the state of the art in robotics and the impact that robots have on the lives of human beings. This thesis has presented a utility-guided framework for motion planning. This general framework for planning has been used to develop algorithms that address two major challenges in real-world planning. Due to its general nature, it may be adapted further to solve the remaining problems that lie between the state-of-the-art and truly autonomous robots interacting with the physical world.

Bibliography

[1] Amato, Nancy, Bayazit, O. B., Dale, L., Jones, C., and Vallejo, D. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*. AK Peters, 1998.

[2] Amato, Nancy M., Dill, Ken A., and Song, Guang. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *Journal of Computational Biology 10*, 3-4 (2003), 239–255.

[3] Amato, Nancy M., and Song, Guang. Using motion planning to study protein folding pathways. *Journal of Computational Biology 9*, 2 (2002), 149–168.

[4] Amato, Nancy M., and Song, Guang. Using motion planning to study protein folding pathways. *Journal of Computational Biology 9* (2002), 149–168.

[5] Apaydin, Mehmet Serkan, Brutlag, Douglas L., Guestrin, Carlos, Hsu, David, Latombe, Jean-Claude, and Varma, Chris. Stochasitc roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. *Journal of Computational Biology 10*, 3-4 (2003), 257–281.

[6] Arno Kamphuis, Julien Pettre, Mark H. Overmars, and Laumond, Jean-Paul. Path finding for the animation of walking characters. In *ACM SIGGRAPH Symposium on Computer Animation* (2005), K. Anjyo and P. Faloutsos, Eds.

[7] Arulampalam, S., Maskell, S., Gordon, N., and Clapp, T. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions of Signal Processing 50* (February 2002), 174–188.

[8] Atkeson, C. G., Moore, A. W., and Schaal, S. Locally weighted learning. *Artificial Intelligence Review 11*, 1-5 (1997), 11–73.

[9] Baginski, Boris. Local motion planning for manipulators based on shrinking and growing geometry models. In *Proceedings of the International Conference on Robotics and Automation* (1996), vol. 4, pp. 3303–3308.

[10] Baginski, Boris. The z^3-method for fast path planning in dynamic environments. In *Proceedings of the IASTED Conference on Applications of Control and Robotics* (1996), pp. 47–52.

[11] Baginski, Boris. Efficient motion planning in high dimensional spaces: The parallelized z^3-method. In *Proc. of 6th International Workshop on Robotics in the Alpe-Adria-Danube Region RAAD'97* (Cassino, June 1997), pp. 247–252.

[12] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence 72*, 1 (1995), 81–138. Special Volume on Computational Research on Interaction and Agency.

[13] Bernoulli, D. Exposition of a new theory of risk. *Econometrica (translated and reprinted)* (1954). originally in Comentarii Academiae Scientiarum Imperialis Petropolitanae, 1738.

[14] Bohlin, Robert, and Kavraki, Lydia E. Path planning using lazy PRM. In *Proceedings of the International Conference on Robotics and Automation* (San Francisco, USA, 2000), vol. 1, pp. 521–528.

[15] Boor, V., Overmars, M.H., and van der Stappen, F. The gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of the International Conference on Robotics and Automation* (1999).

[16] Brock, Oliver, and Khatib, Oussama. High-speed navigation using the global dynamic window approach. In *Proceedings of the International Conference on Robotics and Automation* (Detroit, USA, 1999), vol. 1, pp. 341–346.

[17] Brock, Oliver, and Khatib, Oussama. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research* (2002).

[18] Brooks, Rodney A., and Lozano-Pérez, Tomás. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Conference on Artificial Intelligence* (Karlsruhe, Germany, 1983), pp. 799–906.

[19] Burns, Brandan, and Brock, Oliver. Information theoretic construction of probabilistic roadmaps. In *Proceedings of the International Conference on Intelligent Robots and Systems* (Las Vegas, USA, 2003), pp. 650–655.

[20] Burns, Brendan, and Brock, Oliver. Entropy-guided single-query motion planning. In *Proceedings of the International Conference on Robotics and Automation* (2005).

[21] Burns, Brendan, and Brock, Oliver. Model-based motion planning. In *Proceedings of the International Conference on Robotics and Automation* (2005).

[22] Burns, Brendan, and Brock, Oliver. Toward optimal configuration space sampling. In *Proceedings of the Robotics: Science and Systems Conference* (Cambridge, Massachusetts, 2005).

[23] Canny, John F. *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[24] Canny, John F., and Lin, Ming C. An opportunistic global path planner. *Algorithmica 10* (1993), 102–120.

[25] Cohn, David A., Ghahramani, Zoubin, and Jordan, Michael I. Active learning with statistical methods. *Journal of Artificial Intelligence Research 4* (1996), 129–145.

[26] Connolly, Christopher I., and Grupen, Roderic A. One the applications of harmonic functions to robotics. *Journal of Robotic Systems 10*, 7 (1993), 931–946.

[27] Corporation, The Sony. Aibo. `http://www.sony.net/Products/aibo/`.

[28] Corporation, The Toyota. Intelligent parking assist. `http://www.toyota.co.jp/en/tech/its/program/function/parking.html`.

[29] Cortés, J., Siméon, T., Ruiz de Angulo, V., Guieysse, D., Remaud-Siméon, M., and Tran, V. A path planning approach for computing large-amplitude motions of flexible molecules. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB)* (Detroit, USA, 2005).

[30] D. Conner, H. Choset, A. Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Proceedings of Robotics: Science and Systems* (Cambridge, USA, June 2006).

[31] Dagan, Ido, and Engelson, Sean P. Committee-based sampling for training probabilisitc slassifiers. In *International Conference on Machine Learning* (Catalonia, Spain, 1995).

[32] Flajolet, P., Gardy, D., and Thimonier, L. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Journal of Discrete Applied Mathematics 39*, 3 (1992), 207–229.

[33] Freund, Yoav, Seung, H. Sebastian, Shamir, Eli, and Tishby, Naftali. Selective sampling using the query by committee algorithm. *Machine Learning 28* (1997), 133–168.

[34] Gupta, Kamal. *Practical motion planning in robotics*. John Wiley and Sons, 1998.

[35] Hoffmann, Christoph M. *Geometric and Solid Modeling*. Morgan Kaufmann, 1992.

[36] Holleman, Christopher, and Kavraki, Lydia E. A framework for using the workspace medial axis in PRM planners. In *Proceedings of the International Conference on Robotics and Automation* (San Francisco, USA, 2000), vol. 2, pp. 1408–1413.

[37] Hsu, D., Snchez-Ante, G., and Sun, Z. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *Proceedings of the International Conference on Robotics and Automation* (Barcelona, Spain, 2005).

[38] Hsu, David, Jiang, T., Reif, J., and Sun, Z. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of the International Conference on Robotics and Automation* (2003).

[39] Hsu, David, Kavraki, Lydia E., Latombe, Jean-Claude, Motwani, Rajeev, and Sorkin, Stephen. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*. A K Peters, 1998, pp. 141–154.

[40] Hsu, David, Latombe, Jean-Claude, and Motwani, Rajeev. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications 9*, 4 (1999), 495–512.

[41] Institute, Woods Hole. The jason ii submersible rov. http://www.whoi.edu/marops/vehicles/jason/index.html.

[42] iRobot Corporation, The. Packbot. http://www.irobot.com/government/.

[43] iRobot Corporation, The. Roomba: Robotic floorvac. http://www.irobot.com/consumer/.

[44] Isto, Pekka, and Saha, M. A slicing connection strategy for constructing prms in high-dimensional cspaces. In *Proceedings of the International Conference on Robotics and Automation* (2006).

[45] Jaillet, Leonard, and Simeon, Thiery. A PRM-based motion planner for dynamically changing environments. In *Proceedings of the International Conference on Intelligent Robots and Systems* (2004).

[46] Jaillet, Leonard, Yershova, Anna, LaValle, Steven, and Simeon, Theirry. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *Proceedings of the International Conference on Intelligent Robots and Systems* (2005).

[47] Jensen, Niels Erik. Introduction to bernoullian utility theory. *Swedish Journal of Economics* (1967), 163–183.

[48] Kaelbling, L., Cassandra, A., and Kurien, J. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of the International Conference on Intelligent Robots and Systems* (1996).

[49] Kalisiak, Maciej, and van de Panne, Michiel. Rrt-blossom: Rrt with a local flood-fill behavior. In *Proceedings of the International Conference on Robotics and Automation* (Orlando, FL, May 2006).

[50] Kalman, Rudolph Emil. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering 82*, Series D (1960), 35–45.

[51] Kavraki, Lydia. *Random Networks in Configuration Space for Fast Motion Planning*. PhD thesis, Stanford University, 1994.

[52] Kavraki, Lydia E., Švestka, Peter, Latombe, Jean-Claude, and Overmars, Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[53] Khatib, Oussama. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research 5*, 1 (1986), 90–98.

[54] Kim, Jongwoo, Esposito, Joel M., and Kumar, Vijay. An RRT-based algorithm for testing and validating multi-robot controllers. In *Proceedings of the Robotics: Science and Systems Conference* (2005).

[55] Koditschek, D. E. Exact robot navigation by means of potential functions: Some topological considerations. In *Proceedings of the International Conference on Robotics and Automation* (1987), pp. 1–6.

[56] Kuffner, James J., and LaValle, Steven M. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the International Conference on Robotics and Automation* (San Francisco, USA, 2000), vol. 2, pp. 995–1001.

[57] Kwon, W.H. *Receding Horizon Control: Model Predictive Control for State Models*. Spring, London, December 2005.

[58] Ladd, Andrew, and Kavraki, Lydia. Using motion planning for knot untangling. *IJRR 23*, 7-8 (July-August 2004), 797–808.

[59] Latombe, Jean-Claude. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[60] Laumond, Jean-Paul, and Siméon, T. Notes on visibility roadmaps and path planning. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (2000).

[61] LaValle, Steven M. Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. TR 98-11, Iowa State University, 1998.

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

[63] LaValle, Steven M., and Branicky, M. S. On the relationship between classical grid search and probabilistic roadmaps. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (Nice, France, 2002).

[64] LaValle, Steven M., and Kuffner, James J. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (2000), pp. 293–308.

[65] Leven, Peter, and Hutchinson, Seth. Toward real-time path planning in changing environments. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (2000).

[66] Lewis, David D., and Gale, William A. A sequential algorithm for training text classifiers. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)* (1994).

[67] Lien, J.M., Thomas, S.L., and Amato, N.M. A general framework for sampling on the medial axis of the free space. In *Proceedings of the International Conference on Robotics and Automation* (2003).

[68] Lindemann, Steve, and LaValle, Steven. Incrementally reducing dispersion by increasing voronoi bias in rrts. In *Proceedings of the International Conference on Robotics and Automation* (2004).

[69] Linden, A., and Weber, F. Implementing inner drive by competence reflection. In *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior* (1992).

[70] Lingelbach, Frank. *Path Planning using Probabilistic Cell Decomposition.* PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2005.

[71] Lozano-Pérez, Tomás. Spatial planning: A configuration space approach. *IEEE Transactions on Computers C-32*, 2 (1983), 108–120.

[72] Mazer, Emmanuel, Ahuactzin, Juan Manuel, and Bessière, Pierre. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research* (November 1998), 295–316.

[73] McCallum, Andrew, and Nigam, Kamal. Employing EM and pool-based active learning for text classification. In *International Conference on Machine Learning* (Madison, USA, 1998).

[74] Missiuro, Patricya, and Roy, Nick. Adapting probabilistic roadmaps to handle uncertain maps. In *Proceedings of the International Conference on Robotics and Automation* (2006).

[75] Mitchell, Tom M. *Machine Learning.* McGraw-Hill, 1997.

[76] Morales, Marco, Tapia, Lydia, Pearce, Roger, Rodriguez, Samuel, and Amato, Nancy. A machine learning approach for feature-sensitive motion planning. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics* (2004).

[77] NASA/JPL. Mars exploration rover mission. http://marsrovers.jpl.nasa.gov/home/.

[78] Nielsen, Christian L., and Kavraki, Lydia E. A two level fuzzy PRM for manipulation planning. In *Proceedings of the International Conference on Intelligent Robots and Systems* (Takamatsu, Japan, 2000), pp. 1716–1722.

[79] Nieuwenhuisen, D, Kamphuis, A., and Overmars, M. High quality navigation in computer games. http://www.cs.uu.nl/centers/give/movie/viewPublicPublications.php, 2006. Submitted to Science of Computer Programming.

[80] Nilsson, Nils J. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the International Joint Conference on Artificial Intelligence* (1969), pp. 509–520.

[81] Oriolo, Giuseppe, Vendittelli, Marilena, Freda, Luigi, and Troso, Giulio. The srt method: Randomized strategies for exploration. In *Proceedings of the International Conference on Robotics and Automation* (New Orleans, USA, April 2004), pp. 4688–4694.

[82] Overmars, Mark. A random approach to motion planning. Tech. Rep. RUU-CS-92-32, Utrecht University, October 1992.

[83] Phillips, J.M., Bedrossian, N., and Kavraki, L.E. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *Proceedings of the International Conference on Robotics and Automation* (New Orleans, LA, April 2004).

[84] Plaku, Erion, Bekris, Kostas E., Chen, Brian Y., Ladd, Andrew M., and Kavraki, Lydia E. Sampling-based roadmap of trees for parrallel motion planning. *ITRO 21*, 4 (August 2005 2005).

[85] Randlov, J., Barto, A.G., and Rosenstein, M.T. Combining reinforcement learning with a local control algorithm. In *International Conference on Machine Learning* (2000), pp. 775–782.

[86] Reif, J. H. Complexity of the mover's problem and generalizations. In *Proceedings of the Symposium on Foundations of Computer Science* (1979), pp. 421–427.

[87] Rodriguez, Samuel, Tang, Xinyu, Lien, Jyh-Ming, and Amato, Nancy M. An obstacle-based rapidly-exploring random tree. In *Proceedings of the International Conference on Robotics and Automation* (Orlando, FL, May 2006).

[88] Roy, Nicholas, and McCallum, Andrew. Toward optimal active learning through sampling estimation of error reduction. In *International Conference on Machine Learning* (Williams College, USA, 2001).

[89] Russell, Stuart, and Norvig, Peter. *Artificial Intelligence: A Modern Approach*, 2nd edition ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[90] S. Lindemann, S. LaValle. Computing smooth feedback plans over cylindrical algebraic decompositions. In *Proceedings of Robotics: Science and Systems* (Cambridge, USA, June 2006).

[91] Saar-Tsechansky, Maytal, and Provost, Foster. Active sampling for class probability esitmation and ranking. *Machine Learning* (2003). In press.

[92] Saha, Mitul, Latome, Jean-Claude, Chang, Yu-Chi, and Prinz, Friedrich. Finding narrow passages with probabilistic roadmaps: the small-step retraction method. *Autonomous Robots* (2006). To Appear.

[93] Sánchez, Gilardo, and Latombe, Jean-Claude. On delaying collision checking in prm planning: Application to multi-robot coordination. *International Journal of Robotics Research 21*, 1 (2002), 5–26.

[94] Schölkopf, Berhard, and Smola, Alexander J. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond.* Adaptive Computation and Machine Learning. MIT Press, 2002.

[95] School, The New. History of economic thought. `http://cepa.newschool.edu/het/`.

[96] Schwartz, J. T., and Sharir, M. On the 'piano movers' problem: I. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics 36* (1983), 345–398.

[97] Shannon, Claude E. A mathemtatical theory of communication. *Bell System Technical Journal 27* (July 1948), 379–423.

[98] Shannon, Claude E. A mathemtatical theory of communication. *Bell System Technical Journal 27* (October 1948), 623–656.

[99] Simeon, T., Laumond, J-P., and Nissoux, C. Visibility-based probabilistic roadmaps for motion planning. In *Proceedings of the International Conference on Intelligent Robots and Systems* (1999).

[100] Smola, Alexander J., and Schölkopf, Bernhard. A tutorial on support vector regression. Tech. Rep. NC2-TR-1998-030, GMD, 1998.

[101] Sukharev, A. G. Optimal strategies of the search for an extremum. *U.S.S.R. Computational Mathematics and Mathematical Physics 11*, 4 (1971), 119–137. Translated from Russian, *Zh. Vychisl. Mt. i Mat. Fiz.*, 11(4):910-924.

[102] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[103] Thrun, S., and K.Möller. Active exploration in dynamic environments. In *Advances in Neural Information Processing Systems 4*, J. Moody et al., Ed. Morgan Kaufman, 1992.

[104] Thrun, Sebastian, Burgard, Wolfram, and Fox, Dieter. *Probabilistic Robitics*. MIT Press, Cambridge, Massachusetts, 2005.

[105] Tong, Simon, and Koller, Daphne. Support vector machine active learning with applications to text classifiction. *Journal of Machine Learning Research 2* (2001), 45–66.

[106] Urmson, Chris, and Simmons, Reid. Approaches for heuristically biasing rrt growth. In *Proceedings of the International Conference on Intelligent Robots and Systems* (2003).

[107] Vallejo, Daniel, Remmler, Ian, and Amato, Nancy M. An adaptive framework for single shot motion planning: A self-tuning system for rigid and articulated robots. In *Proceedings of the International Conference on Robotics and Automation* (Seoul, Korea, 2001), pp. 21–26.

[108] van den Berg, J.P., Ferguson, D., and Kuffner, J.J. Anytime path planning and replanning in dynamic environments. In *Proceedings of the International Conference on Robotics and Automation* (2006).

[109] van den Berg, Jur P., and Overmars, Mark. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics 21*, 5 (October 2005), 885–897.

[110] Vapnik, V. *Estimation of Dependencies based on Empirical Data*. Springer Verlag, 1982.

[111] Vapnik, V. *Statistical Learning Theory*. Wiley, 1998.

[112] von Neumann, J., and Morgenstern, O. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

[113] Švestka, Peter. *Robot Motion Planning using Probabilistic Roadmaps*. PhD thesis, University of Utrecht, 1997.

[114] Y. Yang, O. Brock. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In *Proceedings of Robotics: Science and Systems* (Cambridge, USA, June 2006).

[115] Yang, Y., and Brock, O. Adapting the sampling distribution in prm planners based on an approximated medial axis. In *Proceedings of the International Conference on Robotics and Automation* (2004).

[116] Yang, Y., and Brock, O. Efficient motion planning based on disassembly. In *Proceedings of the Robotics: Science and Systems Conference* (2005).

[117] Yershova, Anna, Jaillet, Leonard, Simeon, Theiry, and LaValle, Steven M. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings of the International Conference on Robotics and Automation* (2005).

[118] Yu, Yong, and Gupta, Kamal. An information theoretic approach to viewpoint planning for motion planning of eye-in-hand systems. In *Proceedings of the International Symposium on Industrial Robotics* (2000).