On the Fundamental Relationships Among Path Planning Alternatives

Ross A. Knepper

CMU-RI-TR-11-19

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

June 2011

Thesis Committee:
Matthew Mason, Chair
Alonzo Kelly
Siddhartha Srinivasa
Emilio Frazzoli (Massachusetts Institute of Technology)

For my parents, Mark and Cathy Knepper, who gave me the gift of light.

Abstract

Robotic motion planning aspires to match the ease and efficiency with which humans move through and interact with their environment. Yet state of the art robotic planners fall short of human abilities; they are slower in computation, and the results are often of lower quality. One stumbling block in traditional motion planning is that points and paths are often considered in isolation. Many planners fail to recognize that substantial shared information exists among path alternatives. Exploitation of the geometric and topological relationships among path alternatives can therefore lead to increased efficiency and competency. These benefits include: better-informed path sampling, dramatically faster collision checking, and a deeper understanding of the trade-offs in path selection.

In path sampling, the principle of locality is introduced as a basis for constructing an adaptive, probabilistic, geometric model to influence the selection of paths for collision test. Recognizing that collision testing consumes a sizable majority of planning time and that only collision-free paths provide value in selecting a path to execute on the robot, this model provides a significant increase in efficiency by circumventing collision testing paths that can be predicted to collide with obstacles.

In the area of collision testing, an equivalence relation termed local path equivalence, is employed to discover when the work of testing a path has been previously performed. The swept volumes of adjoining path alternatives frequently overlap, implying that a continuum of intermediate paths exists as well. By recognizing such neighboring paths with related shapes and outcomes, up to 90% of paths may be tested implicitly in experiments, bypassing the traditional, expensive collision test and delivering a net 300% boost in collision test performance. Local path equivalence may also be applied to the path selection problem in order to recognize higher-level navigation options and make smarter choices. This thesis presents theoretical and experimental results in each of these three areas, as well as inspiration on the connections to how humans reason about moving through spaces.

Epigraph

So the shortest day came, and the year died,
And everywhere down the centuries of the snow-white world
Came people singing, dancing,
To drive the dark away.
They lighted candles in the winter trees;
They hung their homes with evergreen,
They burned beseeching fires all night long
To keep the year alive.
And when the new year's sunshine blazed awake
They shouted, revelling.
Through all the frosty ages you can hear them
Echoing, behind us — listen!
All the long echoes sing the same delight
This shortest day
As promise wakens in the sleeping land.
They carol, feast, give thanks,
And dearly love their friends, and hope for peace.
And so do we, here, now,
This year, and every year.
Welcome Yule!

Susan Cooper (*The Shortest Day*, 1974)

The value of teaching without words and accomplishing
without action is understood by few in the world.

Lao Tzu (*Tao Te Ching*, 6th century BCE)

Acknowledgments

Through research and discovery, we strive to illuminate the world with new knowledge. Since Prometheus, this quest for light, both literal and metaphorical, binds one generation to another through all the ages. Darkness inspires fear in us at a primitive level, so we each strive to light the way both for ourselves and for the next generation. I am indebted to the many mentors, role models, and peers who have helped to light the way for me.

Among those who illuminated my path, three stand out for the intensity and duration of that illumination. Foremost, I owe my growth both as a researcher and as a person to their willingness to overcome their own instinctive fear of the darkness by holding back from actively lighting my path when the occasion called for it.

The first two of these are my parents, Mark and Cathy. Thanks to them most of all for teaching me to believe in myself and for never letting me forget the joy of curiosity and learning. They provided everything I needed growing up, and yet some of their greatest acts of love are the actions they did not take. They gave me room to grow into a self-confident adult by recognizing that experience is the best teacher of all. It cannot be easy to watch somebody you love make preventable mistakes. Non-action can be simultaneously the most difficult and the most loving action of all. Every time I messed up, I felt loved and respected – but never judged – by them. As a consequence, I learned, and continue to learn, to light my own way without depending on others. Without this, nothing else would have been possible.

The third major illuminator in my life is my advisor, Matt Mason. Thanks to Matt for being so cool, fun, easy to work with, brilliant, insightful, and modest. Matt embodies the "less is more" philosophy. A few words from Matt in an advising meeting often taught me more than an entire lecture in any of my classes. He often drew my attention to a dark corner rather than illuminating it for me to see. This self-restraint reveals both love and respect. My subsequent actions to try to illuminate that dark corner, whether or not I ultimately succeeded, taught me how to perform original research. That work was hard and often daunting, but Matt gave me space to feel safe in exploring and taking risks. Several of those risks turned into this thesis, which truly would not have been possible without Matt.

Anyone who knows Matt also knows Jean Harpley, who keeps Matt's office and lab running like clockwork—all while balancing about a dozen other miscellaneous roles within the Robotics Institute. I especially appreciate both Matt's and Jean's efforts to insulate me from distractions so I can concentrate on doing great research. I hope it shows in this thesis.

I shared working quarters with a number of fellow students during my time here. All have been great friends and colleagues. Thanks for your support, Tom Howard, Dave Bradley, Jean-Francois Lalonde, Henry Kang, Matt Tesch, and Laura Lindzey.

As a member of the Manipulation Lab, I have shared ideas, paper reviews, and lunch every Friday with Amir Degani, Alberto Rodriguez, Siyuan Feng, Robbie Paolini, and many others. You have all been great!

To the other RI professors I have collaborated with throughout the years: Howie Choset, Al Kelly, James Kuffner, Sidd Srinivasa, Drew Bagnell, and Martial Hebert. I have learned much from each of you, and I appreciate the diverse perspectives gained from exposure to different research topics, styles, and approaches.

I have had a particularly strong collaboration with the Personal Robotics Lab of Sidd Srinivasa. Mike vande Weghe, Dmitry Berenson, Mehmet Doğar, Alvaro Collet, Anca Dr̆agan, and Kyle Strabala: I have have enjoyed being both your peer and your friend. Thanks for treating me as a member of your group!

Academic conferences, aside from being a venue to learn, exchange problems and ideas, and network, are also an opportunity for a little adventure. To my fellow adventurers, Tom Howard, Mihail Pivtoraiko, Mike Stilman, Joe Djugash, Lars and Kristen Erickson, and many others: thanks for all the fun times!

Thanks to the prospective Robotics Institute Ph.D. students I've hosted—Matt Tesch, Anca Dr̆agan, Ben Eckart, and Greydon Foil—for allowing me to give back to the RI and to have a legacy of sorts here at CMU. I appreciate getting to put my house and its spare bedrooms to good use once in a while. And finally, thanks all of you for accepting the RI's admission offer and giving me a perfect hosting record!

Finally, many thanks to Suzanne Muth and Reid Simmons for keeping me, and the whole graduate program, running smoothly. I appreciate the opportunities to give back to the department through hosting prospectives and serving on committees. It's great to be in a department where feedback visibly gets incorporated into the process.

Contents

List of Figures

List of Tables

List of Algorithms

List of Definitions

Chapter 1

Introduction

Robotic motion planning is an exercise in organizing information. Given a costmap (or obstacle map), a set of rules about how the robot moves through space, and a pair of state sets labeled *start* and *goal*, the planning problem is to discover a motion sequence that moves the robot from a start state to a goal state while avoiding obstacles and obeying kinodynamic constraints. The planning problem is usually formulated as either a discrete combinatorial search or a continuous optimization problem.

In either case, the preponderance of motion planning computation is spent in reinterpreting information. Given the solution, represented as a sequence of control inputs, we may verify its correctness by performing a path test. In searching for a solution, path tests consume a significant majority of CPU time. Path tests are slow because the information they require is stored in a relatively inaccessible form in the costmap or obstacle map, and the path testing process coverts it to an immediately useful form reflecting the goodness of the candidate path.

The largest share of computation is consumed in path testing, which comes in two forms, depending on the application. In collision testing, the planner must determine whether a given prospective path for the robot to follow would be likely to result in a collision with obstacles. In this formulation, every robot state has a binary status—it is either free or in collision with an obstacle. A related problem occurs in real-valued costmaps, where cost comprises an abstraction of the risk induced by a particular workspace location. Note that each robot configuration comprises a range of workspace locations occupied by the robot. The planner must evaluate the desirability of a given path, for example by convolving the shape of the robot with the shape of the path. Such an expression gives the risk associated with following that path.

As the number of paths to be evaluated increases, the effort expended by conventional path test algorithms scales linearly. Although these paths represent unique trajectories in the configuration space, their swept volumes in the workspace increasingly overlap. Thus, as the number of paths increases, the work required to test them all individually requires increasing redundancy. By recognizing that all of these paths inhabit a shared workspace, it is possible to improve path testing performance by recognizing and reusing shared information.

In fact, latent shared information may be exploited throughout the motion planning process. Fig. 1.1 depicts the flow of information through a robot in the typical feedback loop. The diagram reveals details of the planner, which is split into three stages: path sampling, path testing, and path selection. This thesis describes opportunities and approaches for recognizing and reusing

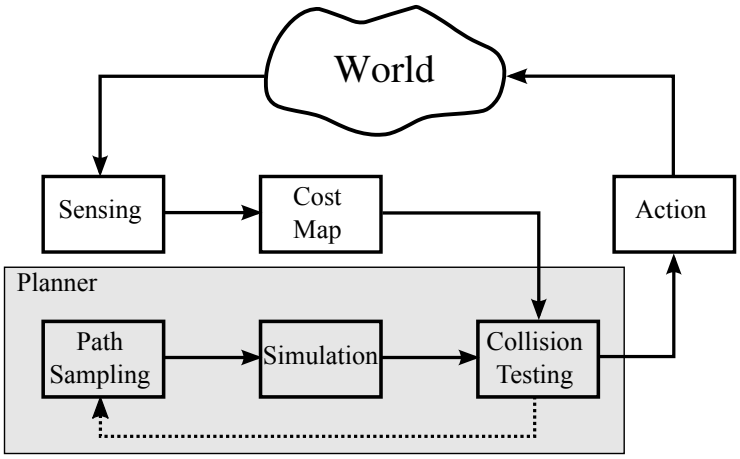


Figure 1.1 Typical data flow within a robot closes the loop around the sense-plan-act cycle, but the planner itself runs open-loop.

information at each stage of the motion planning process.

1.1 Choice of Planner

In discussing information reuse in motion planning, it is helpful to adopt a specific planning paradigm in order to clarify concepts. We focus on the hierarchical planner because we believe it is the simplest case that elucidates the interesting information reuse properties that are the subject of this thesis.

Model predictive planning and hierarchical planning are two techniques that work in concert to plan paths through complex, cluttered terrain over great distances. Predictive models map a robot's control inputs into state space responses, which can be numerically integrated to generate paths in the workspace. Such a process produces high fidelity paths that are inherently feasible to execute on a real robot without post-processing, while incorporating the best possible understanding of both the system and disturbances to it. A model predictive planner typically samples from the control space. Resulting paths depend on both the sampled control trajectory and the initial state of the robot. A collection of input trajectories coupled with resulting workspace paths is referred to as a *path set*. Before it is eligible to be executed on the real robot, each member of the path set must be vetted by testing it for collision in simulation, and some planners require other steps such as computing a cost based on a costmap or equivalent path integral.

The drawback of model predictive planners is that robot models are expensive to simulate. Such planners therefore compute only partial plans, which necessitate frequent replanning. Computational cost and a time budget combine to impose a limited horizon on the model predictive planner, which is also known as a local planner. Such limited range makes the planner susceptible to getting stuck in features of the global topology, such as cul-de-sacs, in which the local gradient, as generated by the potential field method [79], leads into a dead end. To compensate for this shortcoming, a second, global planner may be introduced to provide information on global connectivity of the freespace. The two planners, along with the numerical glue (heuristic

function) to make them work together, form a hierarchical planner.

In the context of a hierarchical planner, the local planner dedicates itself to finding the immediate future path. By replanning continuously while executing the previous planned path, the robot can simultaneously plan and execute with minimal startup delay. It is logical for the planner to spend the majority of its time concentrating on the local area around the robot for several reasons:

1. Sensors give the highest quality data in the immediate neighborhood of the robot.
2. At any given time, dynamics highly constrain the robot's position over the next several seconds (while over longer time scales, many more positions are achievable), so the most care must be taken to avoid collision over the next several seconds, when the problem is hardest.
3. Dynamic obstacles farther away have more time to reconfigure themselves before the robot reaches them.

1.2 Path Diversity

To maximize the probability of survival (and of making progress), it is important to sample control trajectories that maximize the diversity of the resulting workspace paths. Let $\mathcal{O}^n$ be the joint configuration space of all obstacles in the neighborhood of the robot and $\mathcal{O}^n_{free}(\mathcal{P}) \subset \mathcal{O}^n$ be the set of obstacle configurations for which at least one path in path set $\mathcal{P}$ is collision-free. A path set $\mathcal{P}$ maximizes path diversity when the measure of $\mathcal{O}^n_{free}(\mathcal{P})$ is maximized. Basically, paths are the tools by which the planner makes progress towards the goal, and the planner can be most successful when it has the greatest possible variety of viable options on how to proceed.

Robot models can be arbitrarily complex, and so the mapping between control trajectories and paths in the workspace is similarly baroque. Consequently, few model predictive planners depend on diversity in control space when trying to provide diversity in paths. Instead, it is common to generate path sets offline by precomputing each path in a large set and carefully selecting a subset of paths to maximize particular diversity properties. Another approach to generating diverse path sets is to rely on the properties of uniform random sequences to blindly select a set of inputs that will be diverse in the workspace without explicitly precomputing the paths.

1.3 Contributions

In this thesis, we improve upon each stage of the motion planning process using careful information management, both online and offline. We perform diverse, adaptive path sampling in real time by feeding back the results of earlier collision tests. We then move on to accelerate collision testing by recognizing when parts of other paths' collision tests have already tested a new path, thus giving path test results for free. Finally, we improve path selection by recognizing groups of paths possessing shared outcomes.

1.4 Overview

This thesis is organized in two parts. In Part I: Background, we introduce many applications, problems, and prior work solutions. Chapter 2, describes the operation of the hierarchical planner in detail. Chapter 3 establishes a collection of robotic applications for the hierarchical planner and for the information reuse techniques we describe. Chapter 4 delves into a theoretical analysis of the issues affecting local-area motion planning with path sets. Chapter 5 examines the practical aspects of the state of the art in path set design for local planners.

In Part II: The Stages of Motion Planning, we delve into some approaches to information reuse throughout the motion planning process. Chapter 6 explores issues of informed realtime path sampling: specifically, we look at which path is best to sample next, given the collision test outcomes of earlier paths. Chapter 7 deals with an equivalence relation on local paths for use in collision testing. Chapter 8 looks at applying this equivalence relation to route selection. Chapter 9 discusses opportunities for extensions and future work in areas covered by this thesis. Finally, we summarize and conclude in Chapter 10.

Part I

Background

Chapter 2

Hierarchical Planning

The planners described in this thesis represent perhaps the simplest interesting example in which to employ the ideas presented in later sections. Hierarchical planners employ two or more sub-planners (layers) that lie on the spectrum from short-range and high-fidelity (local planner) to long-range and low fidelity (global planner). By combining diverse planners, we may take advantage of sensor information at short range to plan detailed trajectories for obstacle avoidance without spending undue computation planning in far away parts of the environment, about which little may be known.

Hierarchical planners afford a number of advantages.

1. **reactivity:** The local planner replans frequently, meaning that it is reactive to an unpredictably changing environment.
2. **minimal startup latency:** After each replan step completes, the local planner issues a command to the robot for execution during the subsequent planning cycle. Therefore, the robot incurs a minimal amount of lag before it starts moving.
3. **scalability:** The division of labor allows the planner to find paths over great distances.
4. **feasibility:** The local planner also generate detailed paths which avoid obstacles while obeying motion constraints.

Few planners offer this combination of features. For instance, the RRT planner of LaValle and Kuffner [96] is highly scalable and sometimes produces feasible paths, however it is not reactive to a changing environment and has significant startup latency because the entire plan must be completed before execution begins. By contrast, the potential field planner of Khatib [79] is highly reactive as well as scalable and low-latency, but its paths are not guaranteed feasible, and it can easily become stuck in cul-de-sacs since it lacks global guidance or lookahead capability.

In the following sections, we give more detail about the local and global planners and how they are combined using a heuristic function.

2.1 Motivation

We previously described the value of hierarchical planners in decomposing the problem of planning over long distance for robots with constraints. In refutation of the notion that the task can

be accomplished with a single planner at high fidelity all the way to the goal, Reif and Wang [119] showed that the problem of finding a piecewise constant-curvature arc path all the way to the goal is NP Hard. However, hierarchical planning is also appealing from a design standpoint, and as such has been in use in the AI and robotics communities for decades.

The hierarchical planning architecture we employ here is designed not for eventual success in the worst-case but for rapid success in the majority of cases. The plans generated by this planner begin running on the robot with minimal planning delay and succeed in reaching the goal under normal circumstances.

In conducting this work, we sought to build a planner possessing a degree of greediness, which can generate goal-directed behavior that is both fast and subjectively "good enough". By simultaneously employing a degree of lookahead via trajectories in the model predictive local planner, this planner may avoid getting stuck in any but the deepest cul-de-sacs in the potential field. Note that the length of these trajectories (degree of lookahead) places the hierarchical planner anywhere along a spectrum. At one end, zero lookahead leads to something similar to potential field-based controllers, which merely follow the local potential function gradient and can easily get stuck. At the opposite extreme, letting the local planner plan all the way to the goal results in a complete planner. By selecting lookahead distance, we place the planner within this spectrum.

2.2 Prior Work in Hierarchical Planning

Many projects have employed a hierarchy of heterogeneous planners to provide complementary functionality. One of the first mobile robots to do so was ALV [31, 107], which employed four planners called (in descending order) Mission, Route, Local, and Reflexive. The Route Planner resembles our global planner, while the Reflexive Planner performs the function of our local planner. At the local level, ALV's robot model evaluates a handful of linear workspace trajectories based on robot-terrain interaction in order to select the safest course of action that achieves the navigation objective.

The design of such systems as ALV inspired another outdoor planning architecture in PerceptOR [77], which employs a low-fidelity global planner based on Field D* [38] to provide global guidance to a local planner. This local planner evaluates candidate trajectories, generated by sending sampled constant-input controls through a robot model. These trajectories terminate at various points in the workspace. All collision-free local paths imply a complete path to the goal because D* provides a navigation function. The local replan rate is roughly 10 Hz.

Two of PerceptOR's successor robots, Crusher and Boss [57], employ hierarchical planners in different environments. Crusher concentrates on off-road driving, while Boss, the winner of the 2007 DARPA Urban Grand Challenge, navigates roads. A variety of other Urban Challenge teams also explored the use of hierarchical planners to varying degrees (see [82] for a survey of Urban Challenge local planning solutions).

On the Argo platform, Allen et al. [1] implement a separate heritage of off-road hierarchical planners. This work exhibits a subtle distinction from PerceptOR. While PerceptOR and its kin generate a global navigation function which can be queried from any point, Argo's global planner offers a single path for the local planner to track. Thus, only local paths which terminate on the

global path can be considered.

In a twist on the hierarchical planning concept for planetary rovers, Pivtoraiko and Kelly [108] combine search spaces of varying fidelity and scalability into a unified graph. This planner performs comparably to many hierarchical planners with a single D* query at the expense of additional bookkeeping to manage the world-fixed search graph as the rover traverses its environment.

Laumond et al. [92] describe a hierarchical planner for mobile robots with nonholonomic constraints. Their global planner operates in a graph search space defined by the retraction of the free C-space. Meanwhile, the local planner performs potential-field-style gradient descent based on the Reeds-Shepp Reeds and Shepp [118] metric rather than the typical Euclidean distance metric. The Reeds-Shepp metric describes the length of the shortest path for a car with limited turning radius. Such paths can be synthesized using only straight lines and minimum turning radius arcs.

Another body of work relates to hierarchical planning in indoor human environments. Such spaces differ from outdoor rough terrain in that cost is binary; any robot configuration is either in collision with an obstacle or collision-free. Autere [5] describe a multi-resolution grid-based variant of A*, in which each resolution operates similarly to a hierarchical layer by giving more approximate solutions that are topologically informed. Hyun and Suh [63] find connected spaces in a grid and then hone them into higher quality paths in successive hierarchical layers.

Several other works more closely resemble the type of hierarchical planning architecture described in this thesis. Candido et al. [26] discuss the application of hierarchical planning to humanoids. Since the high degrees of freedom make exhaustive C-Space planning impractical, this work instead employs a look-up table of feasible motions, called motion primitives, which generate useful behaviors when applied in context to a terrain. They build a graph of motion primitives that explore the terrain. After finding a path through the graph connecting the start and goal states, the authors execute the resulting series of motions open loop.

Yang and Brock [146] implement a hierarchical planner loosely resembling our architecture. The global planner generates a probabilistic road map with movable nodes to handle dynamic environments. The objective is to maintain an approximation of the changing topology of the environment within this graph. During execution, a potential field local planner follows these graph edges. Like our planner, theirs is incomplete but functions reliably in practical settings.

In a work by Wang et al. [142], a global planner computes the shortest path in a grid from start to goal. A local planner then tries to follow this path by defining a sub-goal where a circle centered at the current robot position intersects the target path. Potential fields drive the robot toward its sub-goal, thus performing receding horizon control (pure pursuit path tracking [141]) until the robot reaches the final goal. The authors provide results on a real robot. This indoor planner, like the others discussed above, does not incorporate dynamics into the planning process, thus setting up fundamental conflicts between path following speed and quality as well as between motion constraints and obstacle avoidance.

Willow Garage's PR2 platform utilizes a hierarchical planner for its office navigation planner. Marder-Eppstein et al. [100] report on a maturity test billed as the "office marathon." Like our navigation planner, they employ a grid-based global planner that neglects kinodynamic constraints. At the local planning level, the PR2 employs the dynamic window approach of Fox et al. [40].

Handey [98] is a fully integrated system capable of recognizing and manipulating polyhedral parts in clutter. Handey's motion planner is composed of a two-layer hierarchy of a coarse-grained kinematic arm planner and a fine-grained hand controller for executing planned compliant grasping. The intuition for the hierarchy is that while the arm had to merely avoid obstacles, the hand had to reach into tight spaces, make deliberate contact with obstacles, and perform fine manipulation.

Perhaps the earliest application of hierarchical planning to robotics was STRIPS [39], a symbolic planner designed to help robots reason about achieving complex goals requiring many actions. STRIPS works by decomposing goals into subgoals in a hierarchical structure. In order for a sequence of actions to satisfy the goal criteria. each action's postconditions must satisfy the following action's preconditions. Since STRIPS lacks any understanding of the nature of the problem and must solve the whole problem at once, it suffers from exponential complexity in the size of the plan.

Sacerdoti [122] introduces ABSTRIPS, an abstraction-based variant of STRIPS that employs an abstraction hierarchy to deliberately ignore preconditions to simplify the problem. After solving the problem at the most abstract level of detail, each step in the abstract solution generates a subproblem to be solved at a higher level of detail. Thus, the intricate problems need only be addressed as simpler subproblems, thus dramatically increasing problem-solving efficiency.

Nourbakhsh [106] contributes a crisp definition and refinement of the abstraction hierarchy concept and also proposes interleaved planning and execution to further enhance the symbolic planning process. By executing the first subplan as soon as it is ready, the solutions to further subproblems may be based on ground truth as a result of the first execution, in order to accommodate incorrect assumptions and execution errors. Our local planner embraces the concept of interleaved planning and execution, extending it to simultaneous planning and execution. Thus, we both simplify the problem via hierarchical abstraction and also hide the planning computation time by parallelizing it with execution.

A recent contribution in hierarchical control by Girard and Pappas [46] shows that an approximate abstraction of a complex system may be used as a basis for simplifying control. A trajectory controlling the simplified system is an approximate solution to the real system, and it may be used to generate more accurate trajectory to control the real system. By generating several levels of increasing abstraction, this process may be generalized to a multi-level hierarchy. Besides being an example of hierarchical control, this paper is significant because the robot models we employ are inevitably approximations of the true systems. Even so, the end result of employing such robot models is increased controllability of complex real-world systems.

Plaku and Hager [110] devise a two-layer interleaved planning system in which the top layer performs the symbolic reasoning responsible for discovering subgoals to deliver to the motion planner. Meanwhile, the lower level geometric planner performs sample-based planning to construct a control-based motion tree. The symbolic planner's subgoals bias the state sampling distribution that drives the motion planner in order to direct the search towards accomplishing task goals.

Finally, Knepper and Mason [83] perform some analysis of hierarchical planning for navigation in a theoretical and simulation study.

Table 2.1 Comparison of capabilities among arm planning techniques. A check-mark here denotes that a planner is "good enough" at fulfilling a capability. The hierarchical planner is the most capable overall.

Planner	**Complete**	**Handles Dynamic Obstacles**	**Avoids Local Minima**	**Optimal**
Arm controllers	✗	✓	✗	✓
Elastic bands and strips, CHOMP	✗	✗*	✓	✓
RRT, PRM, SBL	✓	✗	✓	✗
Hierarchical Planner	✗	✓	✓	✓

* this capability is only available to a limited extent.

2.3 Comparison to Other Planning Approaches

A few noteworthy planning approaches, compared in Table 2.1, share aspects of hierarchical planners. The table compares approaches using four common planning criteria. The comparisons indicate whether a planner's performance is "good enough" in a certain area. We use this phrase to mean that while an approach may not reach a paramount state of achievement in an area, the performance is quite acceptable in everyday applications.

Potential fields [79] are an important and popular control technique. Potential fields compute an objective function that causes the robot to be attracted by the goal state and repelled by obstacles. Arm controllers use potential fields in a purely greedy approach to planning, so they are very susceptible to local minima.

Elastic bands [114] and elastic strips [17] represent a technique for defining a volume which is free of objects. Maneuvering within this space is basically free, but the volume is a subset of reachable space. The methods differ in that elastic bands exist in the C-space, whereas elastic strips inhabit the workspace. CHOMP [115] is a more sophisticated, modern variant of elastic bands. These methods possess a limited degree of ability to switch between solution homotopy classes.

2.4 Local Planners

The *local planner algorithm* is built on the premise that it is relatively easy to generate a workspace trajectory from a control trajectory using the robot model but hard to solve the inverse problem, which requires inverting the robot model. Consequently, the planner works by generating a set of paths through forward simulation and testing their suitability for execution. The basic local planner algorithm is sketched out in Alg. 1.

Robot safety requires that our robot model provide a high-fidelity simulation capable of predicting the robot's response to candidate actions over the upcoming ground shape. Model predictive planning utilizes such a robot model to select and steer the best trajectory.

Model predictive planning (MPP) generalizes the notion of model predictive control (MPC)

[24, 111] to non-linear, time-varying systems with possibly many local minima. In MPC, the controller repeatedly simulates controls being applied to the system via a model. At each step, the controller numerically solves an optimization problem. In the context of a robot, this optimization is called trajectory generation. For mobile robots, such trajectories (either in the workspace or the C-space) may be uniquely determined [74] or may be optimized according to some objective function [58]. In MPC, the controller typically bounds the lookahead time horizon in order to make the computation tractable.

MPP likewise bounds lookahead time, but it samples a diverse set of controls instead of optimizing a single control. The same objective (or heuristic) function as with MPC is used to pick the winning path for execution. For a sufficiently dense, low dispersion path sampling, the resulting controls from MPC and MPP may closely approximate each other. However, MPP has an advantage in objective functions with many local minima (such as with potential fields in clutter) because it does not rely strictly on gradient descent to find high-quality controls.

Independently of the research leading to this thesis, Dunlap et al. [35] presented Sampling Based Model Predictive Control (SBMPC) within the controls community. The authors present results in both robotics and process control applications. SBMPC functions quite similarly to our local planner, discretizing the space of controls and utilizing the forward model to predict the resulting trajectories. Given the similarities between approaches, the contributions of this thesis thus apply to SBMPC as well for many applications.

Model predictive planners are somewhat costly to execute due to two factors. First, they are limited by the speed and complexity of the accompanying robot model. To improve runtime throughput, it is possible to precompute trajectories and store them in a look-up table (LUT) after discretizing both the initial state and control input. Second, the planner must frequently retest all available paths to ensure responsiveness. A rapid replan rate is important to ensure the most up-to-date sensor data for the upcoming ground patch, and replanning also provides the robot an ability to react to dynamic obstacles in real time.

Together, a rapid replan rate and a costly model constrain the amount of available computation during each replan cycle, which has the effect of bounding some combination of depth and breadth of local planning. Planning depth is enforced by the depth of each candidate local path. Constrained computation induces a tradeoff between depth and breadth—the number of path alternatives that can be considered per unit time. The correct choice in this tradeoff involves many factors, such as expected depth of cul-de-sacs, expected width of narrow corridors, and physical parameters like stopping distance and sensor range. The outcome of this process is a set of safe, feasible paths that may be selected for execution. Section 2.6 provides more detail on the process of selecting one of these paths.

In earlier work utilizing local exploration to effect a global result, Cheng and LaValle [27] modify the RRT planner by keeping track of which local paths have been tried and failed due to collision with obstacles. Feedback from these failures is used to bias selection of future path samples. Kalisiak and van de Panne [69] improve upon this idea with RRT Blossom by expanding the entire path set in parallel, much like we do. They reason that in difficult situations with clutter, Cheng and LaValle's planner will eventually do so anyway, and there are gains to be made in considering all of the paths in parallel. This thesis discusses a variety of such gains, and our results therefore apply directly to RRT Blossom as well.

One such gain is realized by replacing the $Get\_Next\_Path()$ function in Alg. 2 line 3, which

Algorithm 1 Local_Planner_Algorithm(w, x, h, $\mathcal{P}$)

Input: w – a costmap object;
x – initial state;
h – a heuristic function for selecting a path to execute;
$\mathcal{P}$ – a fixed set of paths
Output: Moves the robot to the goal if possible

1: **while** not at goal **and** time not expired **do**
2: $\mathcal{P}_{free} \leftarrow Test\_All\_Paths(w, x, \mathcal{P})$
3: $j \leftarrow h.Best\_Path(x, \mathcal{P}_{free})$
4: $Execute\_Path\_On\_Robot(j)$
5: $x \leftarrow Predict\_Next\_State(x, j)$
6: **end while**

Algorithm 2 $\mathcal{P}_{free} \leftarrow$Test_All_Paths($w$, $\mathcal{P}$)

Input: w – a costmap object;
$\mathcal{P}$ – a fixed set of paths
Output: $\mathcal{P}_{free}$, the set of paths that passed collision test

1: $\mathcal{P}_{free} \leftarrow \emptyset$
2: **while** time not expired **and** untested paths remain **do**
3: $p \leftarrow Get\_Next\_Path(\mathcal{P})$
4: $collision \leftarrow w.Test\_Path(p)$ // $collision$ is boolean
5: **if** not $collision$ **then**
6: $\mathcal{P}_{free} \leftarrow \mathcal{P}_{free} \cup \{p\}$ // non-colliding path set
7: **end if**
8: **end while**
9: **return** $\mathcal{P}_{free}$

has traditionally been quite naive. Most hierarchical planners simply iterate through an uninformed sequence of paths, $\mathcal{P}$, testing the set in a brute force manner. Provided that $\mathcal{P}$ is a diverse path set, this method can be effective at discovering feasible trajectory options. However, there is room for improvement by informing the path sampling process based on feedback from prior sampled paths. We return to this issue in Chapter 6, Path Sampling.

There is a degree of latitude in the nature of the paths that make up $\mathcal{P}$. They can, for example, be trees that branch at the root or at many other nodes. We give a detailed history of the path set types we have investigated in the past in Chapter 5, Path Set Design.

Alg. 2 performs the path test operation on the entire path set. Although the algorithm handles each path in isolation, it can be more efficient to consider them in parallel. Schlegel [126] makes efficient use of obstacle information in collision testing a local path set for a mobile robot. The author discretizes the local area containing the path set in the robot-fixed frame. A priori, the author computes, for each cell, the list of paths for which traversal causes some part of the robot to pass over that cell. During collision test, obstacles from the global cost map are projected onto the local map frame. For each obstacle cell in the local map, all those paths traversing the cell are eliminated from consideration. This approach significantly increases collision-testing

efficiency at the expense of some degree of approximation. We demonstrate in Chapter 7 a topological approach to increasing path-testing efficiency that does not suffer from this cellular approximation.

In testing all these trajectories, a rapid replan rate is important to ensure the most up-to-date sensor data for the current robot state. Rapid replanning also provides the robot an ability to react to a dynamic environment in real time. Together, a rapid replan rate and costly computational requirements constrain the amount of available computation during each replan cycle. Bounded computation introduces a tradeoff between depth of search (the horizon of local planning) and breadth (the number of distinct paths explored).

To compensate for the limited horizon, a hierarchical planner employs one or more coarser planners with different tradeoffs. In general, an entire spectrum of planners may be employed in a hierarchy, where the largest scale may involve no more than a heuristic distance estimate. Each successively lower level in the hierarchy plans to a shorter horizon while more accurately representing actual robot motions. For examples in this thesis, the hierarchy is restricted to two levels, local and global. We discuss the global planner in the following section.

2.5 Global Planners

In solving long-range queries, understanding the topology of the freespace is the most essential task because the robot may otherwise become stuck in a cul-de-sac. Even using a map annotated with the connectivity of freespace, finding the shortest path is not trivial. One common approach for representing topology is the retraction of freespace, in which corridors connecting junctions are represented by graph edges. The graph edges may explicitly follow the Voronoi diagram, as in the work of Choset and Burdick [29], Garber and Lin [43]. The Voronoi diagram may also be used to bias probabilistic sampling, as did Holleman and Kavraki [54], Wilmarth et al. [144], Yang and Brock [147]. Cycles in the graph thus represent obstacles. The representation of freespace as a graph introduces approximate representations of feasible motions that make the path planning problem scalable, although the resulting paths typically are not feasible to directly execute on the robot. A more common approach, which we employ here, is to decompose the workspace into a Cartesian grid. This decomposition is also approximate, but it enables the planner to find a path up to the resolution of the grid discretization.

Once the environment has been reduced to a graph, solving the global planning problem is simply a matter of graph search. A long line of algorithms based on the principle of dynamic programming [8] was proposed to discover the shortest path in a graph. One of the earliest such techniques is Dijkstra's algorithm [33], which performs an exhaustive search of the graph. The algorithm constructs a tree from the graph by iteratively expanding the unexpanded graph node nearest the start node until all nodes have been expanded. Upon expanding the goal node, the tree path leading from start to goal represents the shortest path in the graph.

Approximately a decade later, Hart, et al. presented the A* algorithm [51], which improves upon Dijkstra's algorithm by employing an admissible heuristic function in selecting which node to expand. The heuristic function returns a lower bound on the (unknown) distance or cost remaining to the goal state. Using the heuristic estimate, the algorithm may prune away obviously suboptimal nodes without expanding them, thus dramatically improving search times in large

graphs. This feature, in turn, led to improved scalability for robot navigation problems.

A* search results are predicated on two assumptions: the whole environment is known a priori, and it does not change during navigation. These may both be faulty assumptions for mobile robots sensing their environment while traversing it. To address these difficulties, Stentz proposed D* [130], which dynamically replans as the graph representation of the environment changes. This graph is often represented in 2D by a grid costmap with 4- or 8-way connectivity. As cells change cost, the corresponding edge weights change. D* recomputes only the portion of the path that is affected by this change in cost, including downstream nodes. Since the robot moves through the environment and sensed costmap updates occur in the neighborhood of the current robot location, D* searches backwards from the goal. This algorithm dramatically improves global planner scalability when exploring real environments, although planning backwards from the goal introduces additional difficulties for planning in time, since the arrival time at the goal is unknown while the search proceeds.

Koenig and Likhachev introduced D* Lite [85], a variation on D* that is simpler to implement and more efficient in practice. Subsequently in this work, references to "D*" actually imply a D* Lite implementation.

One further enhancement to D* Lite was introduced by Ferguson and Stentz. Like D*, Field D* [38] searches in a costmap grid, but it does not restrict solution paths to grid discretization. Instead, Field D* interpolates costs between map cells, finding the optimal path from cell edge to cell edge. In addition to providing a more optimal path than D*, Field D* also typically produces paths with gentler angles that are easier for path trackers to follow, although any angles in a path prevent perfect path following at non-negligible velocities.

Most of the above-mentioned algorithms perform heuristic search, meaning that the computational demands of the algorithm depend heavily on the quality of the heuristic estimate. Most real-world robots have nonholonomic constraints preventing them from driving arbitrary paths, but such constraints can be challenging to encode in a heuristic. Another source of search inefficiency derives from the presence of obstacles, which heuristic estimates typically neglect. Dolgov et al. [34] introduce a hybrid heuristic that incorporates both of these factors for a car. Their metric computes two submetrics. The first computes an obstacle-free kinematically-feasible path length similar to Reeds-Shepp [118], while the second computes an obstacle aware grid-path solution using Dijkstra's algorithm. The final value returned by their hybrid heuristic function is simply the maximum of these two values. Such clever heuristics more than make up for their computational expense by expanding fewer nodes in the graph search.

As mentioned above, the local planner has a limited capacity to generate, test, and evaluate model-predictive paths. A robot trying to navigate using only a limited-horizon local planner would run into two difficulties. First, it would have no sense of which direction to move, in a local sense, to make progress toward the goal position. Second, the configurations of obstacles in the world often creates cul-de-sacs. A potential field without such local minima, called a navigation function, is needed to direct the robot toward the goal successfully. Such a navigation function requires knowledge of global topology of the workspace to generate, and so model predictive techniques do not scale well to this task.

Instead, our hierarchical planner employs a simplified robot model lacking the kinodynamic constraints to which the actual robot is subject. We run D* Lite [85] for this task because it rapidly returns answers to navigation function queries, reuses previous results, and permits

movement of the robot and obstacles without recomputing more than necessary. In short, D*'s performance is sufficiently fast that it makes up a negligible part of the overall planner's computation.

In the global planner, the robot is approximated as a point or ball shape. We discretize the workspace into a rectangular grid and run the D* search algorithm to rapidly find the cost of a path from any point in the workspace to the nearest goal position. In searching a grid without regard to robot constraints, we trade off guaranteed trajectory feasibility for computational speed. D* offers additional advantages over other graph search algorithms such as reuse of previously generated results and the ability to alter cell costs due to dynamic obstacles with maximal reuse of prior computations.

The ability to rapidly provide a path to the goal from any point in the free workspace is key to the functioning of the global planner. By querying the end-point of a local path, the global planner provides the continuing path to the goal which is topologically, though not kinodynamically, correct. In Alg. 1, invocation of the local planner occurs within the heuristic function, $h.Best\_Path(R)$.

2.6 Heuristic Functions

The heuristic function is responsible for bringing together all the layers of the hierarchical planner and producing a unified planned path all the way to the goal. The path will only be executed for a short distance, though, since the planner must replan rapidly to provide both reactiveness and sufficient high-fidelity lookahead at all times.

In Alg. 1, the local planner generates a set of local paths which are feasible to execute on the target system. The global planner provides a value function (or "cost-to-go") at the endpoint of each local path. Minimizing the sum of local and global path costs results in goal-seeking behavior, known as goal-directedness. Additional considerations may come into play in selecting a path to execute, such as safety. Paths that come close to obstacles may be penalized, for instance.

The overall heuristic function takes the following form. Given paths p each parametrized as $p : [0, 1] \rightarrow Q$, these paths are scored as

$$c_p(p) = c_{gp}(x(p(1))) + c_{lp}(p) + \alpha^T f(p), \tag{2.1}$$

where q is a robot configuration, $x(q)$ is the corresponding workspace coordinate, $f(q)$ is a vector of features on the robot configuration, and α is a vector of feature weights, which may be hand-tuned or learned using a technique like logistic regression. The global and local path costs are represented by $c_{gp}(x)$ and $c_{lp}(p)$, respectively. Note that the global path is implied by a single state because D* maps every state to a path to the goal. Safety and other such considerations are contained in the implementation of f.

After computing the heuristic function on each path in the surviving path set, the path returned for execution is simply

$$j = \underset{p \in R}{\text{argmin}}\ c_p(p). \tag{2.2}$$

These formulas implement a simple, goal-directed hierarchical planner. Some shortcomings include a tendency to guide the robot very close to obstacles and a preference for needlessly following narrow corridors when they happen to represent the shortest path to the goal. Chapter 8 explores an approach to overcome these shortcomings of the $Best\_Path$ heuristic function.

Chapter 3

Hierarchical Planner Applications

This chapter provides an overview of some applications of hierarchical planning. Our main focus, based on work by Knepper et al. [84], is on mobile manipulation planning. The hierarchical model takes separate hierarchical navigation and manipulation planners and combines them into layers of a meta-hierarchy for mobile manipulation planning.

In the remainder of this chapter, we discuss several applications for the hierarchical planner. Section 3.1 provides an overview of the mobile manipulation planning problem. In Section 3.2, we address indoor navigation of a Segway RMP based robot in an office environment. In Section 3.3, we extend the approach to planning for the Barrett WAM arm. We conclude with Section 3.5 by briefly surveying a few diverse robotic applications, which showcase the abilities of the hierarchical planner.

3.1 The Mobile Manipulation Application

A mature robotic platform, HERB (Fig. 3.5), hosts our planner implementations. HERB comprises a WAM arm mounted on a differential drive Segway RMP base. HERB localizes itself in real time during navigation with less than 5 cm of error. During manipulation tasks, on-board perception systems localize obstacles and target objects relative to the robot within 1 cm, thus ensuring collision-free trajectories.

We have implemented a pair of hierarchical planners on HERB. A 2D planner handles navigation for the Segway base, while a 3D arm planner performs manipulation tasks. These two planners, in turn, form a combined mobile manipulation planner, in which the 2D planner generates a trajectory for the Segway, which acts as global guidance. Due to the limited reach of the arm, the 3D planner considers shorter distances and acts as a local planner. It receives a predicted trajectory from the Segway planner and generates a path which reacts to its predicted motion. We demonstrate this combined functionality in simulation.

We believe that mobile manipulation is an ideal application for hierarchical planning because of its natural variation in level of detail. Before manipulating a faraway object, we must initially focus on driving the mobile base. As the target object gets closer, the configuration of the manipulator arm becomes increasingly important, and more time should therefore be devoted to arm planning in the final stage of the operation. With our hierarchical planning architecture,

Figure 3.1 HERB is a mobile manipulation platform comprising a WAM arm and Segway base.

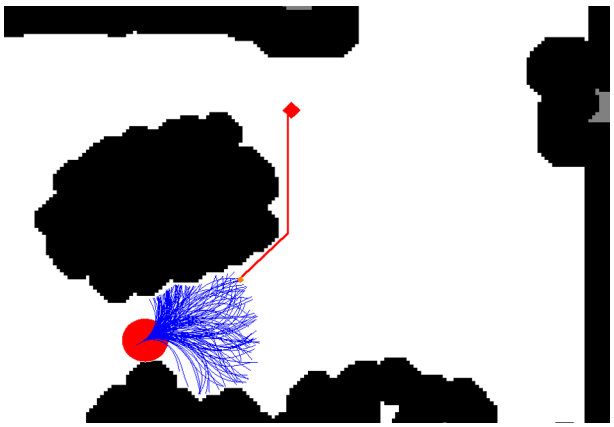


Figure 3.2 Navigation planner view showing local candidate curves in blue and the straighter global D* path (in red) connecting the best local path to the goal (red diamond). Obstacles are in black.

we demonstrate mobile manipulation planning that is spontaneous, effective, and reactive to the environment during execution.

3.2 2D Navigation Planner

In designing a navigation system for HERB's Segway base, we utilized ROS [112]. Just as with the architecture described above, we divided the navigation planner into local and global planners with a heuristic function arbitrating between them. The details of our navigation planner were first described in [83]. The local planner generates a set of polynomial-parametrized curves in action space with a six-second lookahead. In the costmap, each obstacle is dilated by the radius of the robot. Thus, to test a path we merely need to consider the sequence of cells through which it its center passes.

After selecting the best path at each cycle, the planner passes the corresponding command to the robot for open-loop execution. We produced a high-fidelity dynamic model of the Segway based on data collected from HERB driving. We use this model to predict the trajectory resulting from a given control. The predictive model of the Segway is sufficiently accurate as to introduce negligible error over the course of a fraction of a second between commands. Consequently, we do not require a separate closed-loop path-following controller. Instead, we employ a purely reactive controller for safety, plus feedback at the local planner level.

3.2.1 Navigation Planner: Local Planner and Controller

Model predictive planning is executed in a loop. In our system, the local planner replans at 5 Hz, while the controller loops more rapidly at 37.5 Hz. During each cycle, the planner tests a fixed number of trajectories. Paths are represented by polynomial functions of linear and angular velocity controls. We may select the path sequence randomly, in which case we resample the paths during each cycle in order to customize their selection and tune performance. Alternatively,

we also employ a diverse, fixed path sequence as well as other path sampling methods described in Chapter 6.

For long-range navigation, we have found that restricting the set of controls to fixed velocity and limited curvature (i.e. a car-like motion model) produces smoother motion than the more general set of differential drive actions. By contrast, when the robot is near the goal, the full range of differential drive motions becomes crucial to success. Rather than abruptly switching motion types, we sample variably from two distributions on controls (car-like and differential drive) according to goal proximity.

We test each predicted path for collision with obstacles and compute a heuristic cost function on each trajectory. We then pass the lowest-cost control to the controller for execution.

Since we are operating indoors on flat surfaces, the robot model is sufficiently good that we can drive the Segway open-loop for 1/5$^{\text{th}}$ sec at a time. Therefore, the controller's main job is merely to pass commanded velocity trajectories to the robot. However, it also monitors HERB's SICK ladar, broadcasting at 37.5 Hz, for imminent collisions, which are detected by simulating forward the last command beyond the stopping distance. If the controller's "virtual bumper" predicts a collision, then the commanded linear velocity is overridden with zero speed until the hazard no longer obstructs the local path plan. We put the controller infrastructure in place to guarantee highly reactive safe behavior, but the virtual bumper is rarely necessary since the local and global planners will react to obstacle movements at 5 Hz.

3.2.2 Navigation Planner: Global Planner and Heuristic Function

In the navigation planner, D* Lite plans in a map where obstacles are expanded by the radius of the robot. The end point of each local trajectory becomes the starting point for the global path, and the sum of the two path costs (i.e. the line integral of a path with a costmap) gives the total path cost. D* is suitable for use in a real time environment. In an 8-connected grid, our D* implementation expands cells at a rate of 300,000 cells per second, consuming about 0.1 second amortized over a whole navigation run.

The navigation planner's vector of configuration features from (2.1) consists of

$$f_{nav}(\tau) = \begin{bmatrix} c_{ang}(\tau(1)) & c_{prox}(\tau) & c_{sim}(\tau) \end{bmatrix}^T . \tag{3.1}$$

Here, $c_{ang}(q)$ returns the cost resulting from the difference in heading between the terminal configuration in the local path and the gradient of the global navigation function. $c_{prox}(\tau)$ denotes the cost reflecting the proximity of the path culminating in q to any obstacle. Finally, $c_{sim}(\tau)$ reflects the similarity between the commanded action ending at q and the currently-executing action: given two paths of equal cost, we prefer the path that minimizes acceleration.

3.2.3 Experimental Setup

Although this work is motivated by many important applications on real robots, we present many results in simulation in order to establish statistical significance. It is simply not possible to run the thousands of trials required by statistical tests such as Chi-square on a real robot. We submit

the navigation planning problem as a testbed to compare planners and path sets. Within these tests, we utilize a two-level hierarchical planner.

Planning problems consist of a combination of an environment and planning query. Environments were randomly generated within a room measuring twenty meters on a side, in which 10 cm diameter obstacles were randomly positioned until reaching the desired workspace obstacle coverage fraction. Given the size of the robot, coverage densities of up to about 3% are considered solvable.

Planning queries, chosen at random, ask the 0.412 m diameter Nomad Scout robot to navigate from a start to a goal configuration, each chosen randomly so as to be separated by 14 m. Finally, candidate problems were rejected if a 10 cm resolution 8-connected grid planner was unable to solve the planning problem in which obstacles were dilated by the radius of the robot. Note that this grid planner also serves as global guidance to the local planner. Each reported result is an average over a number of runs (between 100 and 1,000) on a fixed set of different planning problems.

At the local level, we simulate a disc-shaped robot of radius 0.206 m, minimum turning radius of about 0.8 m and linear velocity of 0.3 m/s. These numbers are based on experiments from a Nomad Scout robot. Unless otherwise stated, the local planner receives 0.1 sec per replan cycle. In our simulation, we construct path trees with seven discrete curvature actions at each of four depths in the tree. Thus, the densely-sampled tree comprises $7^4 = 2401$ paths. Most other path sets are subsets of the densely-sampled set. The exception is constant-curvature arcs, whose discretization in curvature is as high as required for the requisite number of leaf nodes. Within the tree, each branch lasts for 1.5 s, giving the local planner a total lookahead of 6 s.

Since our test environments are static, we may optionally precompute the global plan from every cell in the free space using brushfire. Since the overhead of D* Lite is negligible as a fraction of overall planning time, this choice does not significantly affect any of our local planning results. At runtime, we utilize a heuristic, *Best_Path*, which merely adds together the times of each local and global path pair (the global path beginning in the cell where the local path terminates), with a penalty term included for the difference in angle between the terminal local path and initial global path.

3.3 3D Arm Planner

In this section, we move to the problem of applying a hierarchical planner to a manipulator arm. We executed our planned trajectories on a WAM arm from Barrett Technology, Inc. with seven degrees of freedom. We utilized the OpenRAVE [32] planning environment, which provides kinematics, collision checking, and visualization capabilities.

3.3.1 The Manipulation Planning Problem

As manipulation represents a huge scope for ongoing research, we concentrate here on the representative task of reaching from a start configuration to any one of a set of goal poses of the end effector. We exploit the affordance offered by the manipulation problem, allowing the hand to

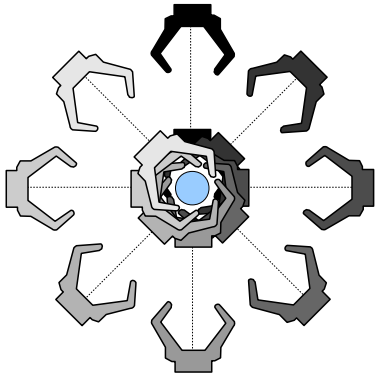


Figure 3.3 Sampling multiple target goal poses around an object. For small finger apertures, the goal state (ready to grasp the object) may be hard to reach. Therefore, we pull back by a small amount and generate a penultimate goal, from which the ultimate goal is reached without search.

approach a target object from any free direction. In fact, we generalize even further by allowing the planner to choose among multiple target objects.

The model we use for goal specification is a simplified form of TSR (Task Space Region [10]) for specifying goal neighborhoods. A TSR describes a range of valid end effector poses. For example, a bottle may be grasped from any side with a small translational tolerance. For this work we kept the implementation simple by instead specifying a small set of goal poses sampled from the TSR. From these goal poses, we compute a corresponding set of inverse kinematic solutions using OpenRAVE's IKfast solver.

Depending on the finger aperture, it may be difficult to find any configuration from which the goal is visible by line-of-sight in joint space. This form of the narrow passage problem is easily solvable because we know that pulling the hand straight back from the target object is a valid motion (pending necessary collision checks). Therefore, by designating a set of penultimate goals set back a short distance from the true goal, a solution becomes easier to find (Fig. 3.3).

3.3.2 Arm Planner: Local Planner

The arm controller relies on a combination of model predictive and PID control. We found this controller to be sufficiently reliable that we could generate purely kinematic plans and count on the controller to execute them accurately.

As with the Segway planner, we allow the arm to begin moving toward the goal before the entire plan has been generated. Planning iterations complete at a rate of about 5 Hz on average, just as on the Segway. Although paths are planned much faster than they can be executed, we cannot replace old commands with new ones as we did in the 2D planner because we currently lack the capabilities both to predict where the arm will be a fraction of a second hence and to interrupt an executing command. Thus, we execute all paths to their endpoints. Fig. 3.4 shows a visualization of all the searched paths through a series of steps culminating in reaching the goal.

During each replan step, the local planner samples a fixed number of straight-line motions in joint space. These paths comprise two categories. To produce a degree of greedy goal-directedness, we generate several paths that move in a straight line in joint space from the current

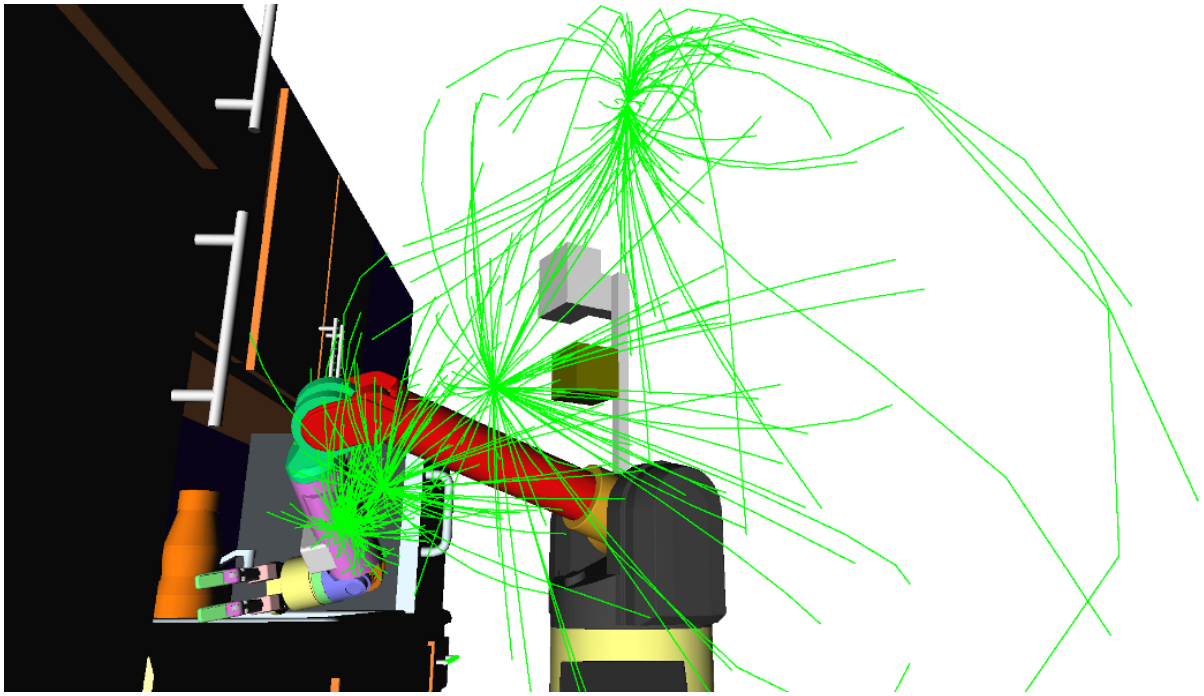


Figure 3.4 An example plan to the goal, showing all considered trajectories. The arm began in singularity at the top of the figure, proceeding down and left toward the counter. It is shown near the final configuration, about to grasp the juice bottle.

configuration to the goal configurations. For the sake of exploration, the remaining motions are sampled from a low-dispersion distribution. We represent all paths as relative motions in joint space, thus they are not particular to a given start configuration.

In the event that one of the goal-directed paths is collision-free, the arm immediately moves to the goal and returns success. However, such goal path collision tests often fail because each target grasp is inherently in close proximity to one of the target objects. In this case, we retain colliding paths by truncating them short of the distance at which the collision occurred (minus 20% of the total path length).

In our initial implementation of the arm planner, we generate the majority of paths by sampling relative motions from a random distribution. Since there is no guarantee that these paths will be useful, we lazily collision test only those paths which are candidates for traversal. If one of the exploratory paths has the lowest cost, then it will be tested for collision with the environment, self-collision with the robot, and exceeded joint limits. A path failing any of these tests is eliminated from consideration.

The path sampling distribution was the subject of some engineering. We break the sampling process into two stages. In stage one, we sample directions in C-space. Then in stage two, we scale each path according to a distribution over lengths that is a function of the remaining distance to the nearest goal. This separation enables the planner to explore widely at the beginning and then gradually move to fine-tuning its position as the arm approaches a solution. We achieve this behavior without fundamentally altering the path generation algorithm at any point during

planning.

We initially tried picking directions by uniformly sampling points from an n-dimensional hypersphere. Such algorithms are somewhat computationally expensive. To improve performance, we next tried sampling from the circumscribing n-hypercube. This fast operation gives a distribution biased toward motions where C-space velocity components are better distributed throughout the joints of the arm (corresponding to the corners of the hypercube). In practice, this distribution appears to better explore the C-space in fewer steps, particularly in cases where the arm starts from a singularity, as in Fig. 3.4

After generating a unit-length vector, we then scale it based on a Gaussian random variable. Both the mean and standard deviation of this distribution are equal to half the C-space distance to the goal, d_i. We scale each candidate path to a joint space length, s_i as

$$s_i \leftarrow trunc\left(N\left(\frac{d_i}{2}, \left(\frac{d_i}{2}\right)^2\right), [0, \infty)\right). \tag{3.2}$$

We apply a heuristic scoring function to all paths and then pass the lowest-cost path that survives the collision test to the arm for execution.

3.3.3 Arm Planner: Global Planner and Heuristic Function

As with the 2D planner, we employ D* Lite to generate a navigation function in 3D workspace position for the end effector, planning in a 26-connected grid. Note that we are treating the hand as if it is floating in space; the resulting D* paths do not reflect potential collisions between the arm and environment. As a consequence of this trade-off, performance is quite fast. In 3D, D* performs over 100,000 expansions per second. Due to the compact workspace, D* expends a mere 1/15$^{\text{th}}$ sec on a typical reaching problem.

The arm planner's vector of configuration features from (2.1) consists of

$$f_{arm}(\tau) = \left[c_{df}(\tau) \quad c_{cs}(\tau(1)) \quad c_{jl}(\tau(1))\right]^T. \tag{3.3}$$

The cost $c_{df}(\tau)$ comes from a squared distance field akin to the signed distance field of Ratliff et al. [115]. This distance field maps each workspace cell to the squared distance to the nearest object in the world. The computation time is negligible (0.06 sec for 1 million voxels on a standard desktop processor). To find a distance field cost, we discretize the arm lengthwise and step through to find the point of nearest contact for the arm. Conveniently, the arm approximates a constant-radius cylinder, so we may readily compute the workspace distance, $d_{ws}(\tau)$. The cost $c_{df}(\tau)$ is computed as

$$c_{df}(\tau) = \frac{1}{d_{ws}(\tau)^2}. \tag{3.4}$$

We measure the straight-line C-space distance to the closest goal configuration, $c_{cs}(q)$. This term may appear redundant since $c_{gp}(x)$ also measures distance to the goal, but the two are complementary. $c_{cs}(q)$ is aware of kinematics but ignorant of obstacles, while $c_{gp}(x)$ incorporates

Table 3.1 Comparison with bidirectional RRT averaged over 5 trials.

Planner	**Average onset of motion (sec)**	**Average time to goal (sec)**	**Average collision checks**
Hierarchical Planner	0.2	12.7	382
CBiRRT	2.4	14.7	1755

obstacles but not kinematics. Since the optimal path is both kinematically feasible and obstacle-free, we use a weighted sum of the two terms. Inspiration for this approach comes from Dolgov et al. [34].

We compute $c_{jl}(q)$ based on the distance of each joint to the nearest joint limit. It is desirable to keep the arm away from joint limits both for manipulability and equipment health. Given the distance d_j of joint j to the nearest limit, the joint limit cost is computed as

$$c_{jl}(q) = \sum_j \frac{1}{d_j^2}. \tag{3.5}$$

3.3.4 Manipulation Planning Performance

We tested the arm planner in a kitchen environment. The task, pictured in Fig. 3.5, is to approach and grasp any one of several bottles on the table. We compared the standalone hierarchical arm planner, running on the real robot, against a state of the art bidirectional RRT-based planner, CBiRRT [9]. CBiRRT is capable of accepting a list of TSRs describing object grasps, hence it is solving a similar problem to our own planner. In performing a comparison, we examined three planning metrics. First, we measured the delay between planning onset and the start of motion. Second, we measured the total task time from start of planning to reaching the goal. Third, we looked at the total number of collision checks each planner performed. Typically, collision checks strongly influence planning time because collision checking is quite expensive. Table 3.1 shows the results of this comparison. The hierarchical planner gives competitive results in all cases—especially in average onset of motion.

3.4 Prior Work in Mobile Manipulation

There have been many approaches to manipulation and mobile manipulation over the years. A recent survey paper by Katz et al. [71] suggests that successful implementations that plan for mobile manipulation in cluttered, unstructured environments generally take on a hierarchical form or otherwise attempt to reduce the complexity of the overall problem by breaking it into simpler subproblems.

Many researchers make a set of simplifying assumptions, which restrict the scope of manipulation planning. Although many diverse manipulation strategies exist, most are difficult to model due to the uncertainties of sliding friction. Thus, researchers typically prefer that robots

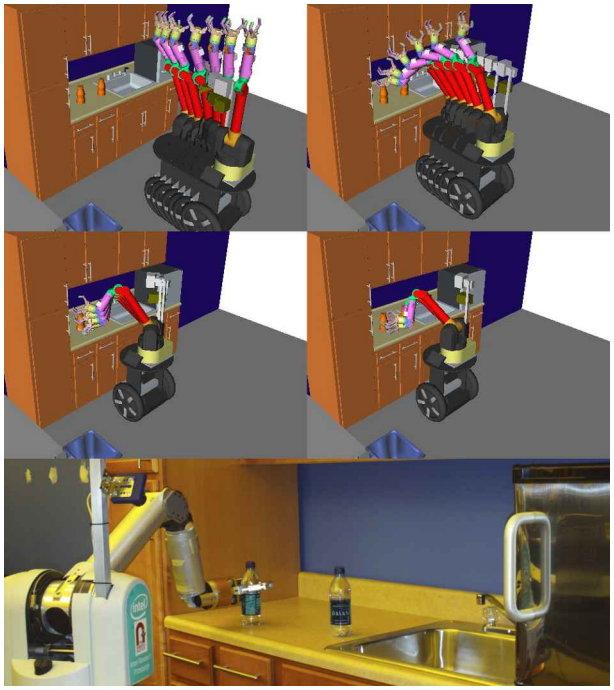


Figure 3.5 HERB is a mobile manipulation platform comprising a WAM arm and Segway base. We demonstrate a solution, based on the hierarchical planning framework, for simultaneously driving toward and reaching for objects. Here we depict such an example task, both in simulation, and on the real robot.

perform an immobilizing grasp on the object under manipulation and otherwise have no contact with it—a technique called pick-and-place. Furthermore, a complex task, such as "set the table for dinner" is generally assumed to be decomposable into a set of subproblems or tasks, each requiring a simpler motion. Taken together, a sequence of such simple tasks can solve the original problem. Koga and Latombe [86] proposed that these tasks be placed in two categories, termed *transit* and *transfer* tasks. A transit path is an arm motion that does not involve an object, while a transfer path moves an object from one stable placement to another. A sequence of tasks generally alternates between transit and transfer actions separated by grasp and release actions. Each transit or transfer task is itself a motion planning problem, while grasping is assumed to be trivial.

Li and Latombe [97] consider a manufacturing scenario in which two robot arms share a workspace. Because parts arrive unpredictably, planning must be done online in real time. The authors devise a set of simplifying strategies involving lower-dimensional search spaces as well as precomputed motion primitives. Using these techniques, the authors were able to produce a working implementation on a pair of real robot arms.

Jain and Kemp [66] focus on a specific mobile manipulation application, the opening of doors and drawers by a robot to assist the disabled. They demonstrate robust opening under incomplete information by applying equilibrium point control, a model of how humans are theorized to perform motor control.

A number of recent papers address the subject of simultaneous task planning and motion planning. In mobile manipulation, the two problems are coupled. In task planning, the sequence

of motions needs to be planned, such as picking up a cup before dropping it off elsewhere. There can be dependencies, such as if a bowl is in the way of the cup. Given a sequence of tasks, a physical motion must be planned for each task, such as approaching the cup or moving the cup to the table. The coupling of task and motion planning occurs because certain motions may be very difficult to plan, while a seemingly more complex series of tasks might actually involve much simpler motions. Thus, by performing task and motion planning simultaneously, a better solution may be realized. Of course, a naive coupling of these planners results in a problem of intractably high dimension.

Ghallab and Ingrand [45] provide a complete mobile robot implementation capable of solving complex tasks. The planner is split between a high-level, symbolic reasoning engine and low-level planning and control systems. In this implementation, a symbolic is generated first, and then the motion plan is attempted. This ordering may cause complications if some symbolic plans do not have feasible motion plans. In such case, the robot must restart planning from scratch.

Nielsen and Kavraki [105] proposed the two-level Fuzzy PRM for solving manipulation problems. The Fuzzy PRM lazily postpones collision testing until a path through the graph has been found. Although it may turn out that this path is invalid, the net result is still fewer paths collision tested, as compared with a traditional PRM. By postponing solving the boundary value problem (BVP), a sub-PRM may be used to connect states in a higher level PRM. The two-level approach permits the authors to decouple the transit and transfer tasks that make up a complex manipulation planning problem.

Siméon et al. [128] introduce topological and geometric formalisms for reasoning about the connection of stable grasps and placements of objects to the transit and transfer tasks that are building blocks of manipulation. They utilize randomized planning techniques to construct a graph joining together C-space manifolds that occupy the intersection of stable grasp and stable placement configurations. Such intersections represent key "bridge" configurations at which an object is stable with or without being grasped.

van den Berg et al. [136] illustrate one type of task planning with their probabilistically complete planner for navigation amongst movable obstacles (NAMO). The NAMO problem is challenging because the robot can only reach a subset of the obstacles, and moving them out of the navigation path may require multiple steps of preparation in which other objects are rearranged to make room for the blocking obstacle. The trick in this case is an abstraction of object configurations such that configurations which induce the same set of connected components of the free space are considered equivalent. By separately modeling the object and robot configuration spaces and abstracting away the robot in planning block motions, the problem becomes tractable.

Two papers by Choi and Amir [28] and Guitton and Farges [50] deal with the question of cross-representation between symbolic and geometric planners. For example, some robot motions result in symbolic changes in the world, such as flipping a switch or opening a door. Utilizing a knowledge base to describe discrete language concepts geometrically, spatial and non-spatial constraints can be rendered in a common language.

Cambon et al. [25] introduce a hybrid planning approach in which symbolic planning states are associated with geometric C-space state sets. Thus, symbolic actions translate to a set of satisfying geometric actions for the robot. Their planning system is powerful, supporting multiple heterogeneous robots constrained both symbolically and geometrically using a general proposi-

tional logic language. However, the system's probabilistic planner does not scale well to complex planning problems.

Hauser and Latombe [52] performs simultaneous task and motion planning by mitigating uncertainty about which tasks may lead to feasible motions. Their planner, I-TMP, distributes search activity for a modified PRM planner across a set of tasks according to the likelihood of success of a given task. Task search is made efficient by searching in a configuration submanifolds induced by constraints placed on the robot. Some care is necessary to sample at the intersection of manifolds that join tasks together. The planner is shown working in simulation for multi-limbed traverse of a rough terrain (i.e. climbing).

Suçan and Kavraki [131] introduce an abstraction called the task motion multigraph, in which each pair of connected states is joined by a set of edges each representing the same motion planning problem posed in various subspaces. For instance, some mobile manipulation problems can be solved by arm motion alone, while others require combined base and arm motion. Lower-dimensional planning solutions are generally simpler to find, but some full-dimensional problems require slow, full-dimensional solutions. Thus, by interleaving computation among various dimensions, the first form of the problem to complete is taken as the overall solution.

Wolfe et al. [145] describe a hierarchical planner for mobile manipulation tasks. They break down a complex task into combinations of simpler tasks in order to achieve a complete task solution. Individual subtasks may invoke motion planners for their subproblems. This process continues until a winning combination of successful motion plans is discovered. Of course, the combinatorics of defining a complete hierarchy of all possible action combinations can become overwhelming. The authors introduce an algorithm called SAHTN to reduce search complexity by recognizing states that are independent of each other and uncoupling them, making dynamic programming more effective.

Kaelbling and Lozano-Perez [68] take the uncoupling of tasks in a planning hierarchy much further. By selectively ignoring task preconditions, they solve parts of the problem independently, even when mild dependencies exist. They then begin executing the partial plans while solving other subtasks. This approach is conditioned on two application-driven assumptions: that subtasks are not intricately coupled, and that there is little penalty for undoing an action that later turns out to be a dead-end. By observing that many common applications, such as household personal robots, possess these properties, the authors are able to solve large, long-horizon problems tractably.

3.5 Other Applications

We have investigated and implemented a variety of other applications besides mobile manipulation. We give a brief overview of several of them here.

3.5.1 Snake Robot Planning

We abstract away the shape and dynamics of a high-degree-of-freedom snake robot in order to discretize its action space into a small set of useful motions that make progress in various directions relative to the body frame. Tesch et al. [133] describe the complexity of executing

motions in such hyperredundant systems. We selected several gaits from that work to constitute discrete actions: rolling left/right, slithering forward (otherwise known as sinus lifting), and left or right in-place rotation based on the sidewinding gait. Although the true actions, being cyclical, can only be stopped or transitioned a certain phases, our approximation permits arbitrary-length application of the actions. By composing these actions, we may construct a path set that in some respects resembles the path sets described above. Rather than attempt to represent the changing shape of the snake, we enclose the entire snake within a bounding box that contains all possible actions, thus simplifying collision testing. A local planner, selecting among combinations of these actions, functions in combination with the global planner built around D* Lite.

3.5.2 The Double Integrator

To demonstrate that the hierarchical planner works equally well with more highly dynamic systems, we implemented a 2D double integrator. A double integrator is a dynamical system in which acceleration inputs to the system are integrated once to obtain velocity and again to get position. We employ this model in simulation for the 2D navigation problem. Given a set of controls that accelerate the robot in assorted directions, the planner selects at each replan cycle a control that produces a trajectory that both avoids obstacles and makes progress towards the goal. Controls take the form

$$u(t) = (\phi(t), \alpha(t)), \tag{3.6}$$

where ϕ is a heading in the plane and α is an acceleration in that direction, both functions parametrized by time. The system model responds to controls by

$$\begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \end{bmatrix} = \alpha(t) \begin{bmatrix} \cos\phi(t) \\ \sin\phi(t) \end{bmatrix}. \tag{3.7}$$

Unlike the snake and Nomad Scout models, which are largely kinematic, the shape of a double integrator path set highly depends on the current state of the system. Thus, it becomes necessary to discretize the state space and create separate lookup tables for a diverse path set and its properties at assorted states.

3.5.3 People Prediction

The hierarchical planner's form permits easy integration with diverse sources of information. For instance, the planner was modified to perform pedestrian avoidance by replacing the global planner with the pedestrian prediction algorithm reported by Ziebart et al. [150]. The prediction algorithm returns a time-indexed value function (a navigation function) in which cost reflects the estimated likelihood that a pedestrian currently being tracked by the robot will be in a particular location at some future given time. By using this value function as a global planner, the robot naturally navigates in such as manner as to avoid blocking or colliding with a person.

Chapter 4

Path Set Theory

Path sets form the basis of our local planner implementation. There has been a limited amount of investigation into the theoretical aspects of path sets. Most of this inquiry has been driven by the desire to generate diverse path sets, which in some way span the space of possible movements subject to some horizon and parametrization. In this chapter, we describe a few basic theoretical tools that we use in later sections to analyze properties of path sets.

4.1 Path Parametrization

Paths can be parametrized in myriad different ways. Two of the most applicable means of expressing the kind of smooth, constrained paths common in robotics are curvature as a function of path length and the pair (*linear velocity, angular velocity*) as a function of time. These parametrizations actually express trajectories in control space. Such a trajectory is taken as an input to the robot model (or as a command to the actual robot), and the output of either system is the true path.

The trajectory commands should be at least piecewise differentiable (piecewise-C^1 continuous). Thus, command discontinuities are permitted following finite intervals. It should also be intuitive and numerically stable. Simple polynomial functions for linear and angular velocity,

$$v(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + \ldots + a_n t^n \tag{4.1}$$

$$\omega(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + \ldots + b_n t^n, \tag{4.2}$$

guarantee smoothness, but the higher-order coefficients exhibit extreme sensitivity for large time-parameter values and can become numerically unstable. Additionally, inspection of a polynomial offers little intuition for the shape of the resulting trajectory.

An alternative control formulation, splines in control space, consists of a series of $n + 1$ uniformly-spaced knotpoints describing linear and angular velocity controls: $(v_0, v_1, \ldots, v_{n+1})$ and $(\omega_0, \omega_1, \ldots, \omega_{n+1})$. There is a unique interpolating polynomial of degree n for each sequence of knotpoints. The knotpoint values are numerically well conditioned and intuitive for sufficiently small values of n.

In cases where we are generating a tree of paths, interpolation does not make sense since one velocity parameter branches into many at the succeeding level. Consequently, we usually turn to piecewise constant curvature commands. These take the form of intuitive n-tuples similar to knotpoints, but the resulting combined controls through the tree are only piecewise C^1 continuous. Nevertheless, the actual followed path is typically subject to higher orders of differentiability due to dynamics.

This response to a commanded action, generated by the robot model, is often not represented explicitly in functional form, but it is instead represented only implicitly as a function

$$p : [0, 1] \rightarrow Q, \tag{4.3}$$

mapping an interval in time to a path in configuration space. In implementation, we represent this path numerically by a series of closely-spaced waypoints in C-space or in the workspace.

Throughout this thesis, we use p to refer to a path and $\mathcal{P}$ to refer to a discrete set of paths. $\mathcal{P} \subset \mathcal{X}$, the continuum path space or a densely-sampled approximation thereof.

4.2 Path Space and Metrics on Paths

Underlying all path set theory is the fundamental issue of how to quantify the difference between paths. We quote the following definition from Munkres [103].

Definition 1 (metric) A **metric** on a set X is a function

$$d : X \times X \rightarrow R$$

having the following properties:

1. (Nonnegativity) $d(x, y) \geq 0$ for all $x, y \in X$; equality holds if and only if $x = y$.
2. (Symmetry) $d(x, y) = d(y, x)$ for all $x, y \in X$.
3. (Triangle inequality) $d(x, y) + d(y, z) \geq d(x, z)$, for all $x, y, z \in X$. □

Definition 2 (path space) The **path space** is a metric space $(\mathcal{X}, \mu)$ in which the metric μ is used to measure the distance between a pair of paths in $\mathcal{X}$. Paths may vary in shape and length. □

The question of which metric to employ on path sets is muddied by the fact that path sets are employed in various different paradigms. Some metrics on path sets that have been tried in the past include:

- approximate area quasimetric[1] of Green and Kelly [48]
- "Exact" area metric
- Hausdorff metric
- Hilbert space L2 metric (also known as bounded variation [70])
- Mutual collision metric
- Survivability metric of Erickson and LaValle [36]

These metrics assume that the paths share a common initial state. Furthermore, each path is parametrized as in (4.3), although some metrics utilize only the position components of the

[1] A *quasimetric* lacks the symmetry property.

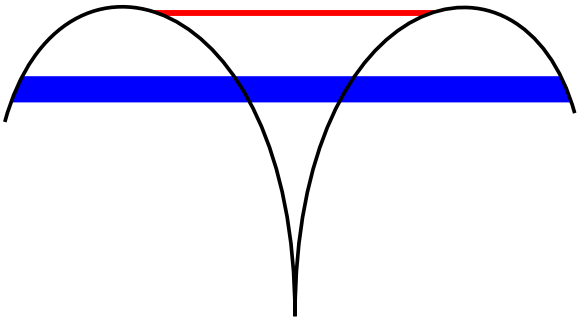


Figure 4.1 Two failures of area-based metrics. A trapezoidal computation of the area between two paths is shown. The red trapezoid at top represents a region in which the space between the two paths is compressed because the paths are nearly parallel. A truer area metric might compute the sweep of the given path points along some geodesic path in path space connecting the two paths (such as by interpolating between the path commands), but such a geodesic is not necessarily uniquely defined. The bottom, blue trapezoid measures an "area" between the paths that would not conventionally be considered to be between them. It is debatable, however, whether the resulting metric value is an over- or under-representation of the difference between the paths.

resulting states. p_i denotes the path i, which is divided into states $p_i(1) \ldots p_i(m)$ that numerically approximate the path.

The Green-Kelly quasimetric approximates the area between paths using rectangular sections:

$$\mu_{gk}(p_i, p_j) = \sum_{k=2}^{m} \mu_{\mathrm{L2}}(p_i(k), p_j(k))\, \mu_{\mathrm{L2}}(p_i(k), p_i(k-1))\,, \tag{4.4}$$

where $\mu_{\mathrm{L2}}(x)$ is the Euclidean distance metric on workspace points. Note that as described in [48] (4.4) is only a quasimetric because it is not necessarily symmetric.

A more exact area between paths can be computed by computing the area of general quadrilateral slices instead of rectangles. However, the idea of using area between paths to represent diversity breaks down for wildly divergent paths. In situations where the direction of path segment intervals $[p_i(\frac{k-1}{m}), p_i(\frac{k}{m})]$ and $[p_j(\frac{k-1}{m}), p_j(\frac{k}{m})]$ are significantly more than 90 degrees different, the quadrilateral between them may include an unrepresentative area that would not ordinarily be deemed to be "between" the curves. Either of these methods may return small values for some quadrilaterals in certain degenerate cases, not properly reflecting the diversity between the two paths. See Fig. 4.1 for two examples.

A common method for finding the maximal separation between two sets is the Hausdorff metric, which returns the greatest distance between any point in one set and its nearest neighbor in the other set. Mathematically, the Hausdorff metric is defined geometrically as

$$\mu_{\mathrm{H}}(p_i, p_j) = \max\{ \max_{1\le k\le m} \min_{1\le g\le m} \mu_{\mathrm{L2}}(p_i(\frac{k}{m}), p_j(\frac{g}{m})),\ \max_{1\le g\le m} \min_{1\le k\le m} \mu_{\mathrm{L2}}(p_i(\frac{k}{m}), p_j(\frac{g}{m}))\,\}\,, \tag{4.5}$$

or topologically as

$$\mu_H(p_i, p_j) = \inf_{\varepsilon}\{p_i \subset (p_j)_\varepsilon \text{ and } p_j \subset (p_i)_\varepsilon\}, \tag{4.6}$$

where $(p)_r$ denotes dilation of p by r: $\{t \in \mathbb{R}^2 \colon \|t_p - t\|_{L2} \leq r \text{ for some point } t_p \in p\}$. In effect, the Hausdorff metric returns the feature with the greatest deviation between two paths. In doing so, it neglects all other features. The lack of sensitivity of this metric to deviations between paths that do not affect their most distant points adversely impacts the use of the Hausdorff metric in applications that might wish to optimize the shape of a path with respect to some functional. On the plus side, the Hausdorff metric gives meaningful results on paths which are arbitrarily similar or diverse.

Another approach is to construct a Hilbert space of dimension md, where m is the number of waypoints in a path approximation, and d is the dimension of the workspace. For each path, the concatenation of its waypoints' workspace coordinates in sequence describes that path as a single point. Let $p_i^c(k)$ denote the coordinate in the c^{th} dimension of the workspace point $p_i(k)$. Furthermore, let the tuple $\gamma(p_i) = (p_i^1(\frac{1}{m}), p_i^2(\frac{1}{m}), \ldots, p_i^d(\frac{1}{m}), p_i^1(\frac{2}{m}), \ldots, \ldots, p_i^d(1))$. It is then sensible to compute the Hilbert L2 metric as

$$\mu_{\text{HL2}}(p_i, p_j) = \sum_{i=1}^{md} \mu_{\text{L2}}(\gamma(p_i), \gamma(p_j))\,. \tag{4.7}$$

As m increases, the approximation of the path improves, and the dimension of the Hilbert space increases. In the limit as m approaches infinity, the space becomes an infinite-dimensional function space. Karaman and Frazzoli [70] refer to this metric taken on the continuum paths as a bounded variation norm.

A final approach to metrics on path space is the mutual collision metric. This metric computes the probability that two paths will collide with the same obstacle. However, the metric has no knowledge of actual obstacles in the world, so it must look at the probability of collision averaged over all possible obstacle configurations. A similar metric, survivability, was proposed by Erickson and LaValle [36].

To compute the mutual collision metric, we suppose that point obstacles are uniformly distributed in the world. The robot is assumed to be a ball of radius r, so each obstacle in C-space is likewise a ball of radius r. These balls are assumed to occupy fraction d_{obs} of the C-space. Any path passing through this space takes the workspace form of a ribbon with rounded ends, known as a *swath*.

Definition 3 (swath) A **swath** is the workspace area of ground or volume of space swept out as the robot traverses a path. □

Each path has some probability of striking an obstacle, which is a function of the swath's area exposed to risk of collision. The area in turn is solely a function of path length; with the exception of very high curvature trajectories (when $1/\kappa < r$), the shape of the path does not affect the swath's area. The formula for the probability of survival—that is, lack of collision between a path's area, A, and an obstacle's area, A_{obs}—can be derived from the Poisson distribution, and it looks like

$$\Pr(survival) = (1 - d_{obs})^{A/A_{obs}}\,. \tag{4.8}$$

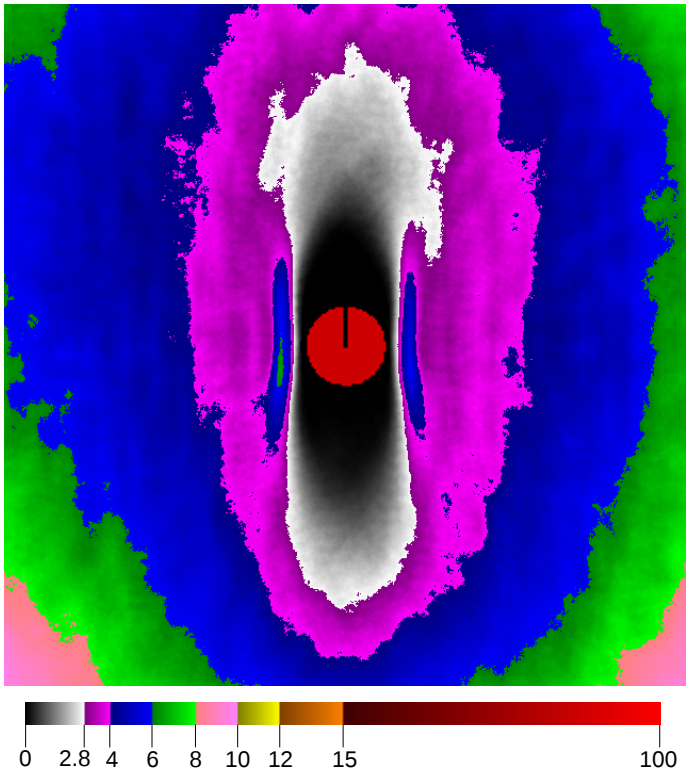


Figure 4.2 Example distribution of obstacles in the frame of a robot-fixed path set local planner. In the world-fixed frame, obstacles are distributed uniformly with a density of 2.8%, so gray regions of the figure represent depressed areas due to active obstacle avoidance, while colored areas represent increased obstacle density. These data were collecting using a path set consisting of 24 uniformly spaced constant-curvature arcs, however, the robot-frame obstacle distribution is subjectively the same with other path sets tested.

Similarly, the probability that two paths collide with a single obstacle depends on the area of their two swaths that is mutually exposed to risk of collision. By letting A represent the area of overlap between a pair of paths, (4.8) may provide a metric based on the probability of mutual collision, hence

$$\mu_{surv}(p_i, p_j) = (1 - d_{obs})^{A(p_i,p_j)/A_{obs}} . \tag{4.9}$$

As with the other metrics, this one comes with several caveats. Of course, obstacles are not typically uniformly distributed in the world, but there is no basis for any better assumption without knowing something about specific environments. Even if obstacles are uniformly distributed in the environment, they may not be uniformly distributed in the robot's planning frame.

Fig. 4.2 shows a uniform obstacle distribution in the world frame as seen from the frame of a robot that is actively performing obstacle avoidance. The effect of obstacle avoidance is to displace obstacles away from the robot as well as the areas immediately in front of and behind the robot. An Ackerman (car-like) path set produced the distribution shown in the figure. For comparison, Fig. 4.3 depicts the statistical average obstacle distribution in those robot config-

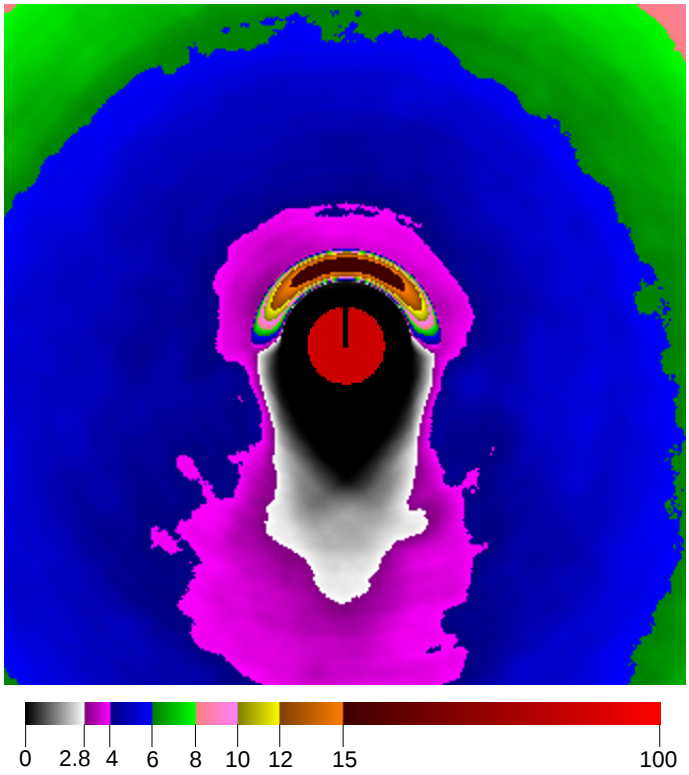


Figure 4.3 As in Fig. 4.2, this figure shows the statistical distribution of obstacles during navigation. However, this snapshot represents only those states which were encountered during path testing but rejected due to collision with an obstacle. The high-probability obstacle region is clearly visible immediately in front of (above) the robot. The regions of elevated probability to the sides of the robot—visible in the previous figure—are absent here because the final step depicted here represents uninformed path exploration. No attempt was made to select paths that would avoid obstacles.

urations where a path tested by the planner collided with some obstacle. Not surprisingly, the highest concentration of obstacle density is immediately in front of the robot.

These distributions could be used to experimentally generate an improved estimate of the probability of mutual collision between pairs of paths in a large path set. Of course, any smaller path set selected on the basis of this metric would then influence the robot-frame obstacle distribution that is used to compute the metric. This chicken-and-egg problem is solved by the fact that the distribution of obstacles empirically varies little among path sets. If necessary, the process could be iterated once or twice to achieve a solid convergence.

Two properties which maximize the uniformity of a distribution of points in a metric space are *dispersion* and *facility dispersion*. Borrowing from Niederreiter [104]:

Definition 4 (dispersion) Given a bounded metric space $(\mathcal{X}, \mu)$ and a set $\mathcal{P} = \{x_1, \ldots, x_n\} \in \mathcal{X}$, the **dispersion** of $\mathcal{P}$ in $\mathcal{X}$ is defined by

$$\delta(\mathcal{P}, \mathcal{X}) = \sup_{x \in \mathcal{X}} \min_{p \in \mathcal{P}} \mu(x, p) \tag{4.10}$$

□

The dispersion of $\mathcal{P}$ in $\mathcal{X}$ equals the radius of the biggest open ball in $\mathcal{X}$ containing no points in $\mathcal{P}$. By minimizing dispersion, we ensure that there are no large voids in path space. Thus, dispersion reveals the quality of $\mathcal{P}$ as an "approximation" of $\mathcal{X}$ because it guarantees that for any $x \in \mathcal{X}$, there is some point $p \in \mathcal{P}$ such that $\mu(x, p) \leq \delta(\mathcal{P}, \mathcal{X})$. Note that all paths in $\mathcal{X}$ are of fixed length and share a start state. This condition is sufficient to assure that $\mathcal{X}$ is bounded for a wide variety of path metrics.

The effect of dispersion on path diversity is to place an *upper bound* on the distance to a point's nearest neighbor. Typically, in implementation the path space $\mathcal{X}$ is represented by a large path set that densely samples the space (i.e. several thousand paths or more). Finding dispersion then amounts to iterating over $\mathcal{X}$ and finding the nearest neighbor in $\mathcal{P}$ at each point. The maximum nearest-neighbor distance will be a close approximation to the true dispersion of the path set.

A related problem is known as facility dispersion.

Definition 5 (facility dispersion) Given a metric space $(\mathcal{X}, \mu)$ and a set $\mathcal{P} = \{x_1, \ldots, x_n\} \in \mathcal{X}$, the **facility dispersion** of $\mathcal{P}$ is

$$\delta_f(\mathcal{P}) = \min_{p_i, p_j \in \mathcal{P}} \{\mu(p_i, p_j)\}. \tag{4.11}$$

□

In order to maximize facility dispersion, one must position k points in a space such that the minimal nearest-neighbor distance within the set is maximized. Facility dispersion's role in path diversity is to place a *lower bound* on the distance to the nearest neighbor of a point in $\mathcal{P}$. For the discrete version of the facility dispersion problem, polynomial-time implementations exist that approach a factor of two of the optimal solution [4, 117]. For a small path set $\mathcal{P}$, computing facility dispersion is significantly faster than computing dispersion since the algorithm only needs to iterate over every point in $\mathcal{P}$ instead of the much larger set $\mathcal{X}$.

If we suppose that the importance of a given path is related to its diversity with respect to other paths in a path set, then both kinds of dispersion have fairly natural interpretations. Dispersion measures the importance of the most important path *not* in the set $\mathcal{P}$, thus suggesting

a one new path that would improve the set's diversity most if only one path could be added. By contrast, facility dispersion reflects the importance of the least important path in the set $\mathcal{P}$. Since each path costs the planner in both time and resources, facility dispersion measures the degree of waste resulting from maintaining relatively similar paths in $\mathcal{P}$. Note that improving either property cannot worsen the other, so they are never in opposition. Thus, an optimal path set would possess both maximal facility dispersion and minimal dispersion.

4.3 Search Space Frame of Reference

In the context of a local planner that replans frequently, there are at least two fundamentally different modes in which a path set may be employed for motion planning. The primary distinction is how the path set moves between replan cycles. As previously stated, all paths in our path set emanate from a common root node. As the robot moves with respect to the workspace, the path set may remain fixed in either the world's frame of reference or the robot's frame. The meaning of the path set is slightly different in the two cases.

With a path set fixed to the world, the paths represent concrete proposed trajectories for the robot to follow. If the path set takes the form of a tree, then the branches represent decision points for the planner. The crucial advantage of world-fixed path sets is that they inherently possess memory of the results of past planning iterations in the form of surviving paths. Assuming that the robot can traverse only a fraction of each local path per replan cycle, and provided that localization and path tracking are adequate, previous results can be reused out to the horizon of the path set. At that horizon, the putative leaves of the tree must be expanded on subsequent replan cycles with further path options to allow forward progress to continue indefinitely. One obvious approach to this problem is to grow a copy of the original path set from each leaf. World-fixed planners may employ a recombinant graph instead of a tree, which accelerates search by combining similar states and eliminating suboptimal routes to any given state. There is a cost to generating or detecting such convergent states in a graph however. Two different approaches to this problem are described by Barraquand and Latombe [7], Pivtoraiko et al. [109].

By contrast, robot-fixed path sets offer only the general idea of a trajectory for the robot to follow. During each replan cycle, the root of a robot-fixed path set (really, of the corresponding control set) moves to the new position of the robot. More precisely, the root of the path set is placed where the moving robot is predicted to be when path set evaluation completes and the next command is issued, a fraction of a second hence. The new path command, when issued, truncates and replaces the old command. Note that for dynamic motion models, the paths' shapes are predicated on the initial state, so the shape of a path set may evolve over time. Since the path set frequently moves by small increments, robot-fixed path sets are capable of searching the environment at finer granularity with the same amount of computation (measured as the number of paths tested per unit time). The trade-off for this benefit is that the planner has no memory of past successful trajectories. For such a memoryless planner to successfully navigate through a corridor, the planner must rediscover at least one safe path through the corridor at each iteration. As the robot traverses a nontrivial corridor, the shape of the corridor changes in the robot frame at each iteration. Consequently, a variety of path shapes will be selected during the course of the corridor's traversal.

One might ask why a deep path set is necessary in the robot-fixed frame context, given that only a small initial interval of any trajectory will ever be executed. One answer is safety; there must exist some dynamically feasible path that extends beyond the safe stopping distance of the robot in order not to risk collision. Another answer relates to sampling theory. While the planner will not end up following the exact trajectory commanded during any iteration, that trajectory posits the existence of a traversable corridor among obstacles. The wider (hence safer) that corridor is, the larger will be the corresponding volume of path space from which future safe trajectories may be sampled. Since a robot-fixed path set must rediscover the corridor many times, this phenomenon introduces a bias toward following wider corridors.

A higher replan rate allows the planner to be more reactive to changing environments. In fact, this planner even explores static environments at a higher resolution than does a world-fixed planner. On the down side, a robot-fixed path set planner lacks memory; it must rediscover a route among obstacles at every iteration or else it will fail to traverse the route.

4.4 Path Diversity

The path diversity problem is the most important theoretical path set question because it is driven by a practical concern: to maximize performance of path set based planners. Briefly, the path diversity problem is to construct a path set containing n paths such that, after testing all paths for collision, the average number of surviving paths across all possible environments is maximized. A related, though different, problem statement is that the probability of at least one path surviving is maximized (so that the robot can still make forward progress).

Given that the configuration of obstacles is not known when designing a path set offline, the problem is similar to that of C-space reachability. The one-and-only safe gap passing through a clutter of obstacles could be located at any configuration, and the robot that cannot achieve that configuration will fail. However, the problem goes beyond reachability because it is insufficient to guarantee that some path in the set reaches every point in space. That path may be blocked by some obstacle prior to reaching the gap. Therefore, the goal of path diversity is to build a path set which contains some path that reaches any given point while avoiding any given set of points. In a finite path set, this is impossible, but one can choose to view each path as approximately representing a corridor of feasible trajectories, all very similar to each other. Thus, the relaxed version of the goal is to find a trajectory to drive the robot within some distance ϵ of a target configuration while avoiding a set of points. Since the goal may be located at any configuration, the problem is therefore to seek the set of paths that approximates any desired path as closely as possible, after culling away some paths that collide with an arbitrary set of obstacles.

To complicate matters, there are multiple tasks in which path set planners are employed. The most significant division occurs between path sets fixed to the world and those fixed to the robot. In the world-fixed case, it appears that diversity is maximized by minimizing dispersion as in (4.10) on the paths in the path set. In the case of robot-fixed path sets, it remains unclear what the correct measure of path diversity is. In the absence of a theoretical answer, we have instead relied on an empirical metric of diversity, computed by measuring the success rate over thousands of simulated trials [82, 83].

4.5 Concerning Completeness and Optimality

Most commonly in the planning community, motion planners are framed in the context of *completeness* and *optimality*. The completeness property states that if a solution exists, the planner will return it, and if no solution exists, the planner returns failure. The optimality property states that no better solution exists than the one returned by the planner.

These properties describe planning ideals, but they both turn out to be prohibitively expensive to guarantee in practice for nontrivial real-world planning problems. Several restrictive forms of completeness have been proposed to make planning problems computable. In deterministic planners, it is common to discretize space, and so *resolution completeness* was proposed to express the notion that a planner becomes complete in the limit as the discretization unit approaches zero.

In more recent years, probabilistic planners (random number based) have come into vogue, and so an alternative form of limited completeness was needed. *Probabilistic completeness* states that in the limit as time (or sample count) approaches infinity, a solution will be found if one exists. Of course, such a probabilistic planner will never return that no solution exists because of the impracticality of exhausting all time. The term sampling-based planning is used to generalize probabilistic planning to also include deterministic sample sequences. In such cases, a planner is said to be resolution complete rather than probabilistically complete, but the same caveats apply.

Until very recently ([70]), sampling-based planners have also sacrificed optimality, which deterministic planners retain only in a limited form that we deem "resolution optimality." In practice, plans returned by most probabilistic planners are highly suboptimal, such that they must be smoothed or otherwise improved as a post-processing step prior to execution.

The generality of sampling-based model predictive planners as presented in this thesis requires that we sacrifice completeness and optimality, but in practice they produce solutions that are "good enough." In effect, these planners trade off completeness and optimality for the ability to plan in large, cluttered, changing environments in real time while obeying kinodynamic constraints.

4.6 Open Questions

Many interesting research questions regarding path sets remain open. We list a few important ones here.

1. Which gives superior performance—and in what circumstances—ground-fixed or robot-fixed path sets?
2. What is the effect of path set size on planning performance?
3. How to measure path diversity of robot-fixed path sets?
4. How to best discretize the action trajectory space for continuous input systems?
5. How to best discretize the action trajectory space for hybrid systems possessing discrete modes, like the snake robot mentioned in Section 3.5.1?

These questions are left as the subject of future research.

Chapter 5

Path Set Design

Having discussed the theoretical aspects of path sets for local planning, we now turn to the problem of constructing a diverse path set. Green and Kelly [48] show that this problem is NP Hard, making it intractable given a large pool of paths from which to choose. Just as with traditional planners, the path sampling algorithms break down into deterministic and probabilistic approaches.

Historically, many local planners restrict their choice of paths to constant curvature arcs of fixed length or time horizon in the robot frame. Some examples include Bohren et al. [14], Buhmann et al. [19], Goldberg et al. [47], Kelly and Stentz [76], Kelly et al. [77], Rauskolb et al. [116], Simmons [129], von Hundelshausen et al. [140]. Arc trajectories have a number of advantages. For an Ackerman steered (car-like) robot, arcs represent a constant command eliciting a constant output. Keeping a control input constant, in turn, means that the next iteration, the same constant control will be available as an option to continue following the arc from the previous iteration. Thus, arcs and other constant actions provide the planner with a modicum of memory, which we address as an important concern in Section 5.3. Additionally, it is easy to sample a set of arcs separated by equal intervals of curvature and bounded on either end by the turning radius of the robot, thus minimizing dispersion (Def. 4) within the subspace of constant-curvature paths. On the downside, arcs place a constraint on the relationship between reachable position and heading that is undesirable for a robot navigating in a cluttered environment.

Several techniques for constructing more general, diverse path sets are introduced below.

5.1 The Green-Kelly Algorithm

Green and Kelly [48] pose the problem of constructing path sets that maximize what they call relative completeness. This algorithm (Alg. 3) takes a deterministic greedy approach to path set construction. To select a path set of size k from a pool of n paths, the authors iterate from 1 to k. After seeding the search with an arbitrary path (the "straight ahead" option), each successive iteration finds and inserts the path which is most distant from the set of paths already picked. This operation is equivalent to computing the dispersion of the path set and returning the center of the largest ball containing no paths in the set. The path at the center of that ball is then added to the path set. In so doing, the algorithm maximizes the facility dispersion of the path set. Polynomial

Algorithm 3 $\mathcal{P}_N \leftarrow$Green_Kelly($\mathcal{X}$, N)

Input: $\mathcal{X}$ – a densely-sampled, low-dispersion path set; $N \leq |\mathcal{X}|$ – the target path set size
Output: path sequence $\mathcal{P}_N$ of size N

1: $\mathcal{P}_0 \leftarrow \emptyset$
2: $n \leftarrow 0$
3: **while** $n < N$ **do**
4: $\quad n \leftarrow n + 1$
5: $\quad p \leftarrow \underset{x \in \mathcal{X}}{\operatorname{argmin}} \left(\delta(\mathcal{P}_{n-1} \cup \{x\}, \mathcal{X})\right)$
6: $\quad \mathcal{P}_n \leftarrow \mathcal{P}_{n-1} \cup \{p\}$
7: **end while**
8: **return** $\mathcal{P}_N$

time solutions to the facility dispersion problem, such as this greedy path selection algorithm, were shown to approach a factor of two of optimal facility dispersion [4, 117].

5.2 Other Deterministic Approaches

Branicky et al. [16] proposed two algorithms for designing diverse path sets of a given path count by selecting paths from a larger path set. These algorithms are based on a discretization of the workspace into a square grid of cells. The authors then represent paths by the set of cells through which they pass. The first algorithm, *Path Diversity Inner-Product*, maintains a set s comprising the union of the cells in each constituent path in its path set. The algorithm begins by seeding the path set with the path occupying the minimal number of workspace cells. It then iteratively adds new paths to the path set by selecting those paths whose intersection with s is minimized. Thus, the overall algorithm seeks to minimize the incidence of multiple paths colliding with one obstacle by greedily building a path set that minimizes workspace overlap.

The second algorithm, *Path Diversity Inclusion-Exclusion*, is named for the principle of inclusion-exclusion, a method of finding the union of a set of sets. In this form, we are interested instead in the probability that a path set (set of sets of cells) is not blocked by obstacles. That value, $\Pr_{pnb}(\mathcal{P})$, works out to the sum of the probabilities of each separate path ($\Pr_{cnb}(p_i)$) not being blocked by an obstacle minus the sum of probabilities that each pairwise union of paths is not blocked by an obstacle, and so on. Mathematically,

$$\begin{aligned}
\Pr_{pnb}(\mathcal{P}) &= \Pr(p_1 \text{ not blocked or } p_2 \text{ not blocked or } \ldots\ p_n \text{ not blocked}) \\
&= \sum_i \Pr_{cnb}(p_i) - \sum_{i \neq j} \Pr_{cnb}(p_i \cup p_j) + \sum_{i \neq j \neq k} \Pr_{cnb}(p_i \cup p_j \cup p_k) - \\
&\qquad \cdots + (-1)^{n-1} \Pr_{cnb}(p_1 \cup p_2 \cup \cdots \cup p_n) \\
&= \sum_{A \in 2^{\mathcal{P}} \setminus \emptyset} (-1)^{|A|-1} \Pr_{cnb}\left(\cup_{p_i \in A} p_i\right) .
\end{aligned} \tag{5.1}$$

The authors compared path survival rates on path sets generated with these two algorithms, a like-sized random path set, and the original full path set across varying obstacle densities. The random and full path sets performed identically, while the Inner-Product algorithm outperformed the others.

Branicky, et al. inspired Erickson and LaValle [36] to propose an alternative formulation of path diversity, which they called survivability, and a path set design algorithm based on the concept. Survivability measures the fraction of paths in a path set T that survive when a single path in T collides with an obstacle, which is allowed to range in size from zero to infinity. First, they define r-survivability, $\alpha(T, r, x) \in [0, 1]$, giving the fraction of paths in T which survive an obstacle of radius r centered at x. Next, the authors define the survivability of a path, p as

$$\sigma_p(T) = \frac{1}{e_p} \int_0^\infty \int_0^{e_p} \alpha(T, r, p(s)) ds dr \,, \tag{5.2}$$

where e_p is the length of path p. Finally, survivability of an entire path set is obtained by summing the survivability of each path as

$$\sigma(T) = \frac{1}{|T|} \sum_{p \in T} \sigma_p(T - p) \,. \tag{5.3}$$

These equations form the basis of a greedy algorithm to select, from a large path set, a subset of paths that maximizes survivability.

The GESTALT navigator of Goldberg et al. [47] runs on the Mars Exploration Rovers (MER), Spirit and Opportunity. GESTALT considers a set of short, constant curvature arcs in the workspace. It scores each according to (a) safety as indicated by a grid costmap, and (b) how well the arc seeks the goal. The current Mars rovers drive at less than 5 cm/sec over rough terrain, so predictive dynamics models are less important than static calculations of wheel slip, rollover, and high centering. The MER rovers did not have a global planner when launched because there are few cul-de-sacs in the Martian environments they were designed for. Long after rovers' arrival on Mars, engineers uploaded a version of Field D* [38].

Another deterministic approach is by Lacaze et al. [88, 89], who contribute a sophisticated local planner, the ego-graph, for mobile robots. The ego-graph is a directed acyclic graph of workspace trajectories originating from the robot's current location—much like the robot-fixed path set tree we describe in Section 4.3. However, the ego-graph's nodes are constrained to be on concentric circles centered on the current robot position. The ego-graph's first level of paths consists of smooth trajectories directly traversable by the robot, while all successive levels of paths in the graph consist of straight lines that serve to approximate future motions. Nodes are sampled most densely around each circle in the direction of travel and quite sparsely behind the robot.

Not unlike our path sets, Frazzoli et al. [42] propose a set of control-parametrized motion primitives for a given dynamical system that fall into two categories. Trim trajectories represent steady-state controls that can be executed in a given state for an arbitrary length of time. Fixed-length maneuvers transition between states and are used to connect trim trajectories together. The Maneuver Automaton is a system for expressing permissible sequences of motion primitives drawn from a finite-sized library. A planner computes the optimal composition of primitives

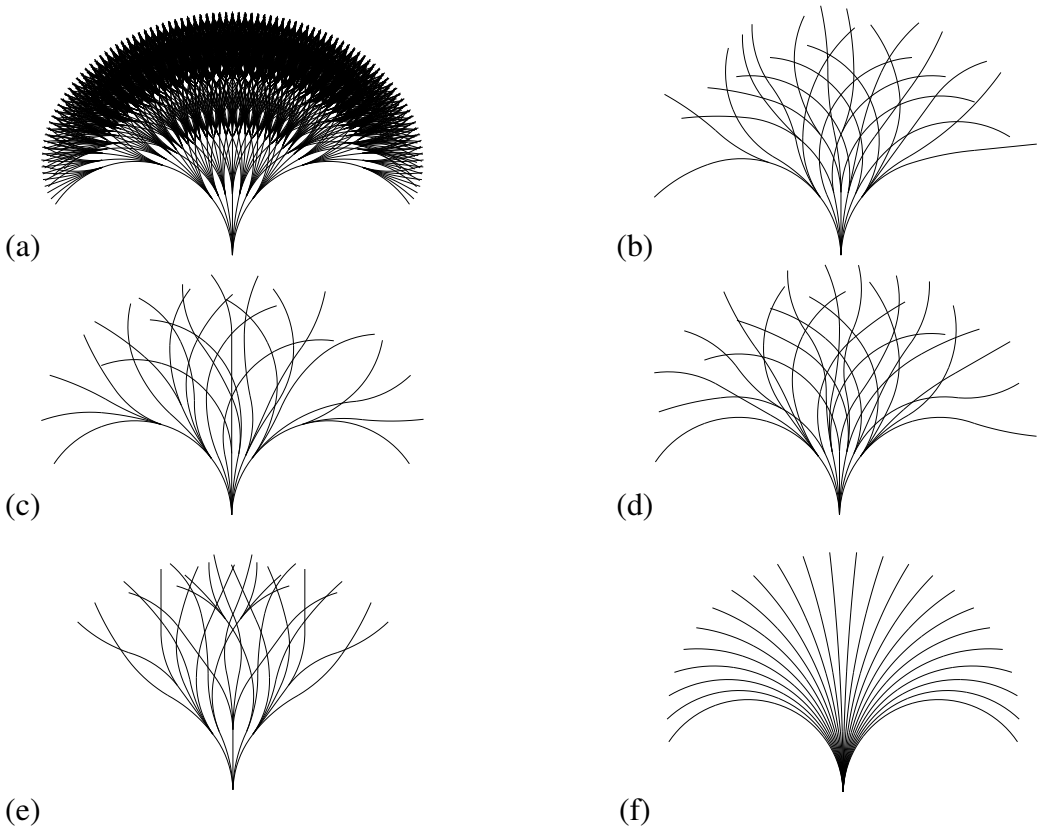


Figure 5.1 Examples of path sets. (a) Full 2,401-path data set; (b) path set generated with a modified Green-Kelly approximate-area metric; (c) path set generated with the Hausdorff metric; (d) path set generated using a mutual-collision metric, which estimates the probability of two paths colliding with the same obstacle; (e) the best performing randomly-generated path set; (f) constant-curvature arcs.

connecting steady-state start and goal states according to some objective function in the absence of obstacles.

5.3 Random Sampling

Given the computational requirements and suboptimality involved in efforts to date to construct minimal-dispersion path sets, it is worth exploring other options. A sequence of uniformly distributed random numbers has low (though not optimal) dispersion and is quite simple to generate. Thus, by relying on randomized sampling, it is feasible to dispense with off-line path set construction and generate path sets in real time. But how do these path sets perform compared to the deterministic algorithms described above?

Knepper and Mason [83] address this question in the context of robot-fixed path set performance. In that work, we show that the Green-Kelly algorithm, which seeks to greedily minimize dispersion, does not return the optimal solution in the robot-fixed context. In our experiments, we generated a large set of random path sets by sampling 24 leaves from a densely-sampled "full" path tree containing 2,401 leaves. Furthermore, we generated a set of 100 random tasks,

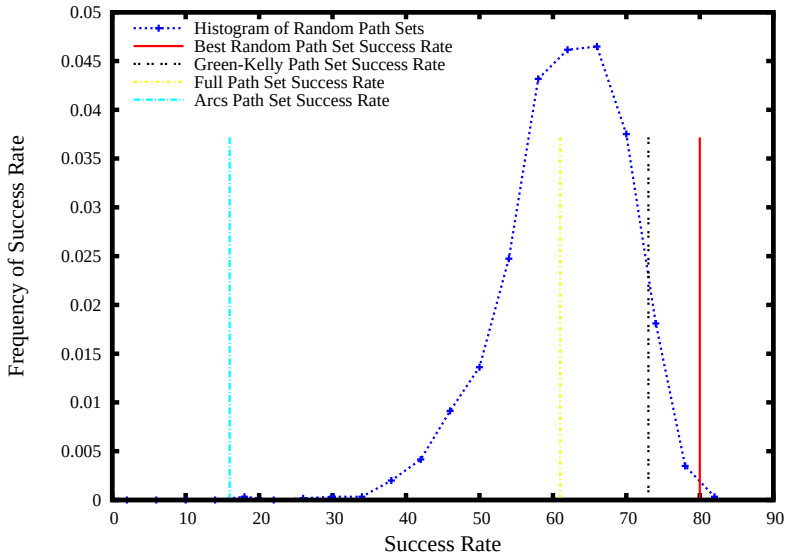


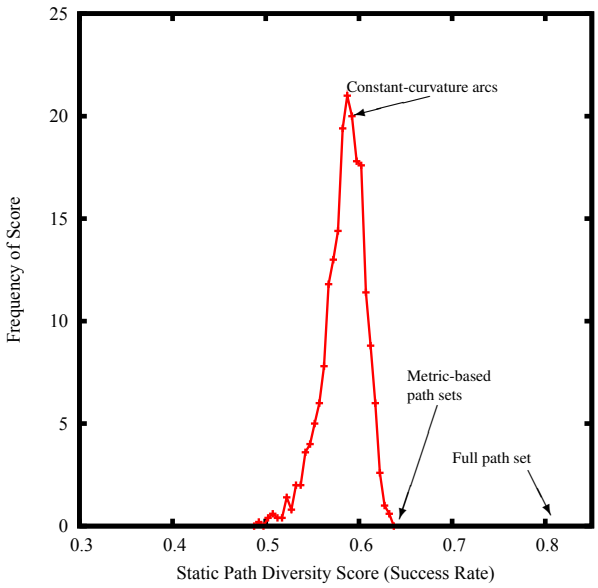
Figure 5.2 Frequency distribution histogram showing success rate of various path sets. A higher success rate indicates a more robust path set in planning experiments.

to which these path sets and some deterministic ones (see Fig. 5.1) were applied. The planner reports success upon reaching the goal state, and it reports failure if no paths survive collision testing at any time during the run. Results (Fig. 5.2) summarize the fraction of tasks in which the path set allowed the planner to reach the goal. The best-performing random path set outperforms the best deterministic path set by approximately 10%.

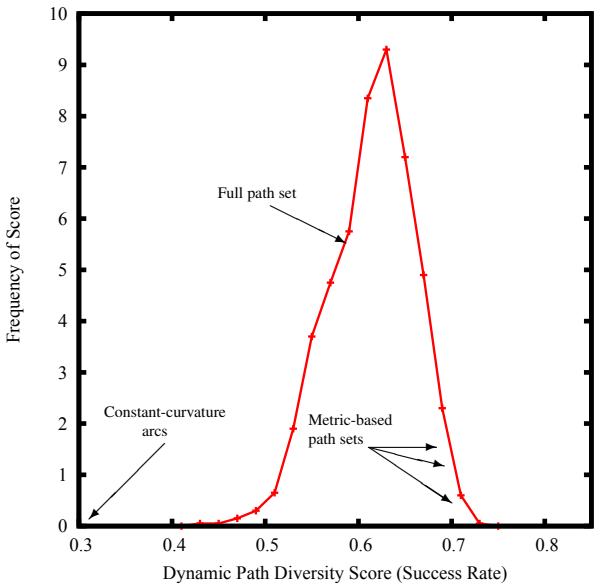
Note the inclusion of the "full" or densely sampled path set in Fig. 5.2. Although path testing consumes much longer than the replan cycle time permits, we allow the full path set in entirety as though it could all be tested. Contrary to intuition, this path set performs worse than many others (even though it contains all of their paths). In short, this phenomenon actually reflects a failing of the heuristic function rather than an innate attribute of the path sets. For a fuller discussion of this phenomenon, which we term *non-monotonicity*, see Section 8.1.

Knepper and Mason [82] subsequently repeat this experiment while distinguishing two cases. In the world-fixed frame case, we refer to the performance as static path diversity. Meanwhile, in the robot-fixed frame, path set performance is called dynamic path diversity. Dynamic path diversity results are as reported in our previous work, while performance in the static path diversity case is quite different (see comparison in Fig. 5.3). There, performance of all the deterministic path sets except constant-curvature arcs exceeds performance of the best random path set. We believe this result indicates that dispersion is a good predictor of world-fixed path set performance, and random paths have higher (worse) dispersion than greedily-constructed deterministic path sets.

Note again the "full" (or densely sampled) path set, which is permitted to cheat by receiving unlimited planning time in order to exhaustively evaluate all 2401 paths. We see strong perfor-



(a) Histogram of static path diversity. This plot shows performance of randomly-generated path sets compared to several deterministic path sets in the world-fixed path set case. The red curve represents a histogram of random path set performance. Other path sets of interest are picked out along the curve. The greedy "metric-based path sets," including Green-Kelly, Hausdorff, and mutual collision, all have virtually the same performance in this test.



(b) Histogram of dynamic path diversity. The curve represents a histogram of the performance scores on robot-fixed path sets. The scores of several special path sets are also shown.

Figure 5.3 Comparison of static and dynamic path diversity. 1000 random path sets were each tested on 1000 dynamic planning problems and scored based on their success rate and completion time.

mance of the full path set in the static case, as the dense sampling is able to find even small gaps between obstacles reliably. By contrast, the full path set performs below average in the dynamic case, reflecting the non-monotonicity property we alluded to previously. Although the full path set still has an advantage at finding narrow gaps, it also possesses the capability to find paths driving very near to obstacles. The heuristic path selection function, which chooses the shortest time path to the goal, will always prefer such paths near obstacles. Driving near obstacles has the effect of drastically reducing available path choices in future replan cycles, thus negating the full path set's advantage. We present a solution to this drawback in Chapter 8.

The most important result of this work, however, is in comparing performance on the same set of random paths between the two reference frames. The correlation in performance on this large set of random paths between the world-fixed and robot-fixed cases is only 0.2175, indicating only a mild relationship between the two cases (Fig. 5.4).

Finally, in order to create a hybrid between the world- and robot-fixed cases, we may employ *continuations*.

Definition 6 (continuation) A **continuation** is the remaining unexecuted portion of the path selected during the previous replan iteration. □

By preserving the selected path as an option for the successive round of path tests, continua-

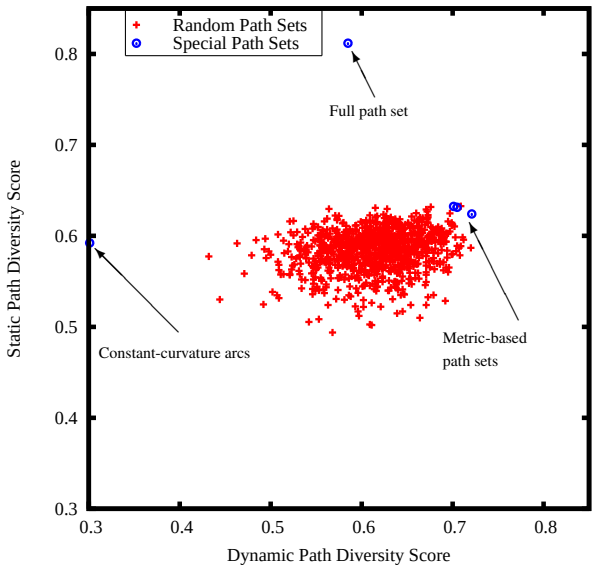


Figure 5.4 Annotated scatter plot of path diversity correlation. Each path set's scores are plotted for the static (vertical axis) and dynamic (horizontal axis) cases. A path set that performs well in both would be at the top-right, while one which performs poorly overall would appear at the bottom-left of the image. The correlation coefficient between these two distributions is 0.2175, reflecting only a very mild connection between performance in these two distinct problems. A strong correlation would appear as a diagonal line from lower left to upper right.

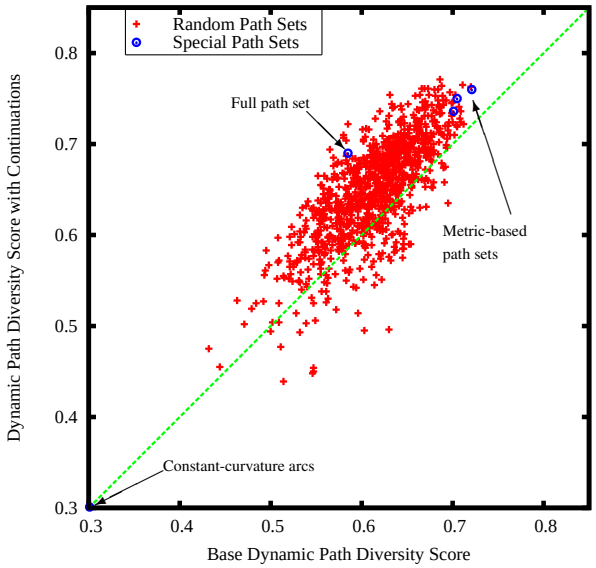


Figure 5.5 The effect of continuations on navigation performance. To better understand the effect of continuations on robot-fixed path set planning performance, each path set was retested with an extra path: the untraversed remainder of the previously-commanded path. Path sets which degrade in performance appear below the diagonal line, while those path sets that improve are shown above the line. About 15% of random path sets are adversely affected by the additional continuation option.

tions add a minimal amount of memory to robot-fixed path sets, and so performance is expected to increase when they are employed. We find that 85% of random path sets and all of the deterministic path sets see a performance increase averaging 5% when using continuations (Fig. 5.5).

One question we gloss over in the above discussion is how to properly generate random path sets. If each path is generated by a separate control from start to finish, then one need only sample randomly in the space of controls. Certain scenarios, however, call for path trees, such as those in Fig. 5.1. The most prominent need for path trees comes in the world-fixed path set case, where branching points provide the planner with decision points.

Sampling of leaves in the path tree, as was done in the two papers above, would appear to be the wrong approach in the world-fixed context because of the non-uniformity in effective branching factor induced by such sampling. Suppose, for example, that in a tree of depth four and maximum branching factor seven, giving 2,401 leaf nodes, 24 nodes are sampled from the leaves. The expected number of branches varies with level according to the following (approximate) formula.

$$\text{Exp}[L_i] = b\left(1 - \left(\frac{b-1}{b}\right)^{\frac{k}{\prod_{j=0}^{i-1} L_j}}\right) \tag{5.4}$$

Here, L_i is the number of branches at a node in the i^{th} level in the tree, b is the maximum number of branches, and k is the number of sampled leaf nodes. Note that we are making an assumption that $k \ll b^d$, with d the depth of the tree. The expected effective branching factors at each level are $L_0 = 6.8269$, $L_1 = 2.9286$, $L_2 = 1.1825$, and $L_3 = 1.0140$. This nonuniformity is significant because of how the world-fixed planner treats expansion at the leaves of the original tree. If a fresh copy of the original tree is rooted at each leaf node, then as the robot moves through the tree, the number of available choices will cycle through the above progression, on average. This nonuniformity of choice is not conducive to effective navigation.

If instead a path set is desired that has a uniform expected branching factor throughout, then we require a nonuniform distribution at the leaves, which can be more easily achieved by sampling each branch from a uniform distribution. Placing a uniform expectation on every node in the tree induces a branching factor of $k^{1/d} = \sqrt[4]{7} \approx 2.21$ throughout the tree for the values above. Assuming that a copy of the tree could be placed at each leaf node to provide a full d levels, a tree sampled as we describe would provide a similar search experience starting from any node, both in terms of the total number of nodes and edges searched, as well as the tree's ability to find gaps and penetrate obstacles. These traits are valuable in a world-fixed frame planner.

One could generate such a tree by first using weighted random samples to select the number of branches at each node according to a Poisson distribution, and then choosing which branches to include from an unweighted random distribution. Such an algorithm will generate a collection of path sets whose leaf count has mean k. The variance of leaf count around this mean occurs as a result that sampling at a given level is a function only of previous levels—not of edges already sampled within the current level. Thus, the distribution can be tightened somewhat by adjusting the expected number of branches at each successive level of the tree to account for the total number of nodes at that level. Given the total number of nodes N_i at level i, the mean value for use in the Poisson formula is

Figure 5.6 Two examples of uniform branching factor random path sets. Each is generated by iterating through levels 0–3, sampling branches according to (5.5) and the accompanying text. Compare to Fig. 5.1(b)–(e), which are examples of path sets sampled uniformly at the leaves.

$$L_i = \left(\frac{k}{N_i} \right)^{\frac{1}{d-i}} . \tag{5.5}$$

Thus, we see considerable opportunity for path set design, even with randomly-generated path sets.

5.4 Dynamic Path Sets

All of the methods for path set design that we have discussed up to this point involve fixing a single path set and using it throughout the duration of the planning task. In some cases, this off-line generation mode is done out of necessity since some design techniques can be slow to execute. Others, such as uniform random leaf sampling, can run in real time. Assuming that it is feasible to generate path sets online during each replan step, is there a benefit to doing so?

In the robot-fixed context, continuously varying the path set at random is advantageous compared to picking a fixed path set at random. However, the best randomly chosen fixed path set consistently outperforms a continuously varying random path set. These results are shown in Fig. 5.7. This result is hardly a surprise since demanding a unique random path set at each cycle is a request for quantity over quality. Most of the varying random path sets, taken as fixed path sets, would individually perform worse than the best random path set tested.

The trouble with continuously varying random path sets is that they are still task agnostic. To realize superior performance, it is necessary to incorporate knowledge of the current situation into the selection of paths during each replan cycle. Inputs to the problem include the configuration of obstacles, current robot state, and actions still available versus those selected during past sampling cycles. To realize performance gains, the resulting path set construction algorithm must save more time through avoiding unnecessary collisions than the overhead of clever path sampling. This problem is the subject of Chapter 6: Path Sampling.

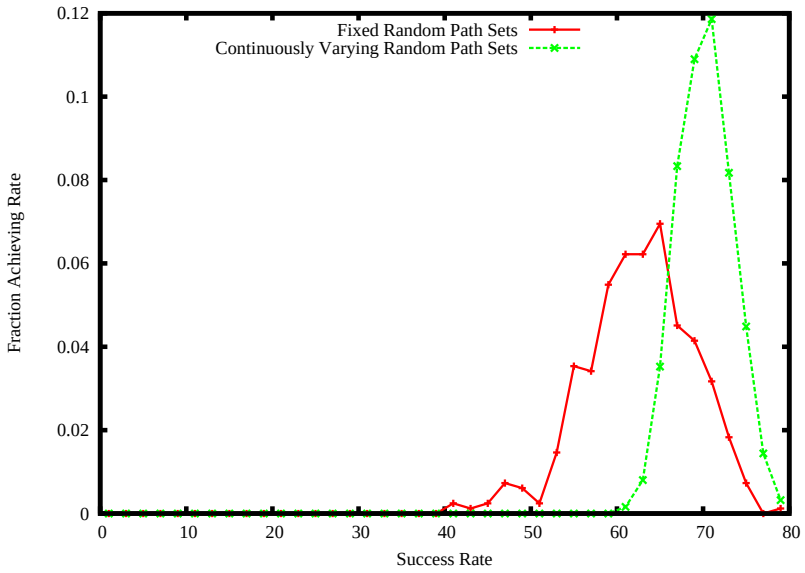


Figure 5.7 Histogram of fixed random path sets versus continuously varying random path sets in the robot-fixed frame. Each fixed random path set is selected once and run on 1000 test problems. For the continuously varying random case, a new random path set is chosen at the outset of each replan cycle. While a user may select the best fixed random path set and expect good performance in other problems, a continuously varying path set's performance is unpredictable, so only the distribution on these path sets is meaningful.

5.5 Prior Work in Path Set Generation

A fundamental work laying out the rationale and assumptions underlying our original local planner is the two part paper by Kelly and Stentz [75, 76]. This paper presents Ranger, a local planner for dynamics-aware planning for mobile robots. The targeted applications fall into two categories where dynamics are a vital component of planning: high-speed navigation and cluttered spaces in which nonholonomic constraints are significant. Ranger tests approximately ten constant speed-and-steering commands per replan cycle. Each arc is fed through a forward robot model, which returns full 3D state plus velocities. These state variables, in turn, are used to evaluate controls for safety and effectiveness in navigation.

Barraquand and Latombe [7] present a planner that samples six actions in the command space—the actions of the Reeds-Shepp car [118]. Unlike Reeds-Shepp, however, the actions are integrated through a robot model to produce short trajectories. Each trajectory is designed to be just long enough to reach an adjacent cell in the C-space grid. Naturally, such a planner generates a tree of motions. To reduce work performed by A* when searching the tree, the planner culls away all but the first path to land in a given C-space cell. In this planner, cost is determined by the number of reversals in a path, while the A* heuristic is zero, making the search identical to Dijkstra's algorithm.

Dunlap et al. [35] generate a search space in the form of a directed acyclic graph by sampling branches at random in control space, integrating the actions forward, and discovering where they terminate. In the manner of Barraquand and Latombe [7], the authors discretize the state space and eliminate redundant paths arriving at equivalent states. Unlike Barraquand however, they use more meaningful costs and heuristics in order to achieve minimal distance paths.

The path set generation problem also arises in computer graphics, where it is used to generate motion graphs—a technique for realistic character motion generation. Kovar et al. [87] were among the first to propose the idea, in which various clips of motion capture data are concatenated to form a tree of continuous motions. Smoothly interpolating between clips presented a challenge in addition to the problems of path diversity and search efficiency explored in this thesis proposal.

In an extension to the motion graph concept, Lau and Kuffner [90] precompute a tree of feasible motions for a character, for example using motion-capture data. Then, using a lookup table, the authors are able to rapidly discover feasible routes between different states subject to the obstacles in the character's environment. For longer-range motions, a two-level planner hierarchy is used to generate subgoals within the range of the precomputed path tree. Reitsma and Pollard [120] extend the work of Lau and Kuffner and others to splice together portions of motions in order to maximize realism of the resulting animation.

5.5.1 State-Sampled Path Sets

Sampling in state space is valuable because it is in the state space that low dispersion leads directly to an ability to avoid obstacles. The inherent challenge is the two-point boundary value problem (BVP) in which states must be connected by a feasible trajectory. Several recent works sample deterministically in the state space and rely on a model-predictive trajectory generator to solve the BVP.

Howard et al. [57] sample at a set of states in the robot frame that are cross-track offsets to a nominal global planned path. An online trajectory generator attempts to join each of the candidate states to the current robot state. The resulting trajectories become a path set tuned for lane-following behavior.

A slightly different approach is the State Lattice of Pivtoraiko et al. [109], in which a trajectory generator is employed offline to find a set of trajectories that join up to create a regular grid or lattice pattern that is tessellated throughout the workspace. At runtime, the world-fixed paths are tested for collision, with A* running on the surviving graph structure to find the shortest path.

5.5.2 Probabilistic Planners

Prior to these recent state space sampling approaches, several randomized state space sampling planners arose as a means of combatting the curse of dimensionality. These planners typically solve the BVP using straight-line interpolation between states. Although kinodynamic versions of some of these planners exist, they have received comparatively little attention.

Kavraki et al. [73] introduced the Probabilistic Roadmap (PRM) as a multi-query planner that explores high-dimensional state spaces by sampling randomly in the state space and then attempting to connect neighboring states to form a graph. To generate a path using the PRM, the start and goal states must first be connected to one of the nodes in the graph and then an A* search may be performed to find the shortest path. Complete visibility coverage of the space by the graph nodes (the Art Museum Problem) is necessary to assure completeness, but this can be difficult to achieve in some cases.

By contrast, LaValle and Kuffner's Rapidly-exploring Random Trees [96], or RRTs, perform single query planning by expanding trees from the start and goal states that grow towards each other. At each iteration, a random state is sampled, and the planner attempts to connect it to its nearest neighbor in one of the trees. If it is able to join to both trees, then a path has been found connecting the start and goal.

The expansive space planner (ESP) of Hsu et al. [59] offers another form of roadmap-based probabilistic planning. In each planner iteration, the algorithm grows a tree by selecting a node at random and growing a new edge by applying a random action from the state represented by that node. If the new edge represents a collision-free motion, then a new child node is added to the tree. Although the ESP has demonstrated some effectiveness on real planning problems, its performance tends to degrade more quickly than that of PRMs and RRTs.

The Single-query Bidirectional Lazy (SBL) planner by Sánchez and Latombe [124] combines elements from both the RPM and RRT approaches. It samples randomly in state space to connect nodes in the graph, but collision checking is postponed until the result is needed, since in the above planners, a majority of collision-tested paths will not be used in the final solution.

Probabilistic planners like PRM, RRT, ESP, and SBL can never be guaranteed to find a solution. Instead, researchers rely on the concept of probabilistic completeness which states that as the number of samples increases without bound, the probability of finding a solution path approaches one. Another factor is more important in practice though. In practice, these planners rapidly find a solution path most of the time. The tradeoff for such rapid path-finding is that the results are suboptimal–sometimes highly so. Therefore, a post-processing "smoothing" step is generally applied to find a shorter collision-free path. Often, the smoothing step takes

longer than planning itself. A side effect of smoothing is that probabilistic planners generally do not come with feasibility guarantees. Since the path uses a pure state space representation, it is left to the path tracker to follow it. An alternative both to highly suboptimal paths and to path smoothing is provided by RRT* [70], which rapidly finds a suboptimal solution and then, as time permits, converges toward a more optimal solution by refining the sampled tree using additional state samples.

One probabilistic planner that generates feasible paths described by a control policy is by Frazzoli et al. [41]. They propose a modification to the RRT algorithm. Like RRT, a state is sampled at random, and the planner tries to join it to an existing node in the tree. Rather than try to connect only the sample's nearest-neighbor node as RRT does, this planner attempts to connect the sample to each node in the tree, one at a time. The nodes are tried in order from nearest to farthest, based on a metric function that is aware of the dynamics. An edge connecting each tree node to the newly-sampled state is generated by utilizing a controller that works in the absence of obstacles. If the resulting edge collides with an obstacle, it is thrown out and the next node is tried. When a collision-free edge is found, it is added to the tree and the iteration step terminates. The tree constructed by this algorithm is described by a set of controls (just as with our planner), so that any path through the tree describes a feasible motion even for highly dynamic systems. The planner is effective at penetrating clutter and permits moving obstacles, provided that their path is known in advance.

5.5.3 Optimizing Planners

Another class of local planners does not build an explicit search space but instead searches the continuum by performing gradient descent optimization.

Khatib introduced potential fields [79], in which an imaginary basin of attraction by the goal state is tempered by repulsion of obstacles. The net "force" on the robot moves it down the center of corridors and toward the goal. Potential fields are not so much a planning technique as a controller because they do not employ lookahead. The value of lookahead is avoidance of local minima in the potential field. This situation arises when the attractive force of the goal is precisely canceled by the repulsion of several obstacles, causing the robot to get stuck.

An alternative optimization approach comes from Ratliff et al. [115]. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) computes an initial guess trajectory and then optimizes it until it becomes free of collision. Optimization is performed based on a signed distance field, which discretizes the environment into cells and computes the shortest distance from each cell to the freespace boundary. CHOMP is not guaranteed to find a collision-free trajectory, although experimental results suggest it works quite well. Because CHOMP operates in the space of trajectories connecting start and goal (or subgoal) states, it avoids local minima in the C-space. CHOMP is instead subject to getting stuck in local minima in trajectory space, for example selecting a trajectory in a suboptimal homotopy class. Although framed as a limitation, such a property can sometimes be advantageous, as we discuss in Chapter 8: Route Selection.

Another sort of continuum planning was introduced by Quinlan and Khatib in the form of elastic bands [114], which represent an alternative solution to the tradeoff between model fidelity and scalability. Given a global C-space path produced by some low-fidelity planner, a series of C-space "bubbles" covering the path are computed by the planner. These bubbles set their radius

according to the distance to the nearest obstacle to the path. By maintaining overlap between consecutive bubbles, an elastic band of guaranteed freespace creates a corridor through which the robot may travel without performing collision checking. The width of the corridor provides some flexibility in which the robot may smooth out corners or comply with other kinodynamic constraints. The name "elastic band" refers to the property that the corridor may be efficiently deformed in real time based on dynamic obstacles.

By analogy to elastic bands, Brock and Khatib propose elastic strips [17], which create "tunnels" of bubbles in the workspace for the robot to move inside. Just as with elastic bands, these tunnels provide some slack for the robot to maneuver while retaining a collision-free guarantee. Both elastic bands and elastic strips are free of local minima, but they have limited capacity to explore different homotopy classes of solution path as the configuration of obstacles evolves.

A combination of deterministic state space sampling and optimization was proposed by Howard [55]. This planner begins with a State Lattice as described by Pivtoraiko et al. [109], and then performs optimization on graph nodes, edges, or both. The resulting planner is far more likely to find a sufficiently low-cost path through clutter without resorting to extremely fine-grained discretization. The resolution only needs to be fine enough to find each local minimum (corridor) of interest.

5.6 Path Set Design

Most motion planners do not explicitly design path sets, instead leaving it up to chance. However, the issues we highlight on this topic are important when studying the performance of all motion planners. We therefore hope that the tools and techniques we discuss in Part I of this thesis find broad application within the community.

We now move on to Part II, in which we examine each stage of the motion planning process, uncovering available information that traditional planners simply throw away. We start with online path set design by feeding back information on path tests to future sampling steps.

Part II

The Stages of Motion Planning

Chapter 6

Path Sampling

The motion planning problem is to find a path (parametrized by arc length) or trajectory (parametrized by time) that guides the robot from a given start state to a given goal state while obeying constraints and avoiding obstacles. In either case, the solution space is high dimensional, so motion planning algorithms typically decompose the problem by searching for a sequence of shorter, local paths, which solve the original motion planning problem when concatenated.

Each local path comprising this concatenated solution must obey motion constraints and avoid obstacles and hazards in the environment. Many alternate local paths may be considered for each component, so planners select a combination of local paths that optimizes some objective function. In order to generate such a set of candidate paths, the planner must generate many more candidate paths, each of which must be verified against motion constraints and collision-tested prior to consideration. Motion planners generate this large collection of paths by sampling—most often at random or from a low-dispersion sequence as defined by Def. 4.

All the information needed to find collision-free path samples exists within the costmap, but the expensive collision-testing process prevents that information from being readily available to the planner. A negative collision-test result (i.e. no collision) offers reasonable certainty to the planner that executing a particular path will not harm the robot or its surroundings. In contrast, a positive collision-test result is typically thrown away because the path is not viable for execution. Such planners may subsequently waste time sampling and testing other paths that collide with the same obstacle.

This policy of discarding information about colliding paths highlights a major inefficiency, which especially impacts realtime planning. Every detected collision provides a known obstacle location. This observation may not seem significant at first, as all detected collisions represent obstacles already stored in the costmap. However, not all such obstacles are equally relevant to a given local planning problem, and so we can benefit by storing relevant costmap obstacles in a form more immediately available to the planner. We argue that the planner may derive increased performance by feeding back the set of collision points, known from prior collision tests, to the path sampling process, as in Fig. 6.1.

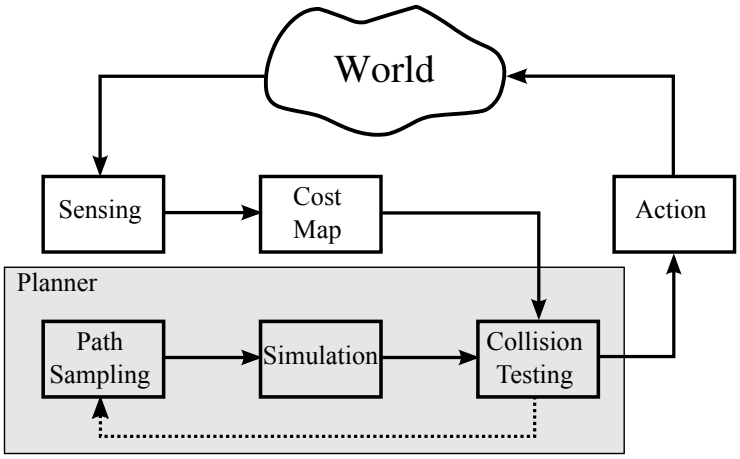


Figure 6.1 Typical data flow within a robot closes the loop around the sense-plan-act cycle, but the planner itself runs open-loop. In this chapter, we close the planning loop by informing path sampling based on the results of collision-testing earlier paths against obstacles.

6.1 Path Sampling and Path Parametrization

The general path sampling problem is to supply a sequence of distinct paths $\{p_1, p_2, \dots\} = \mathcal{P} \subset \mathcal{X}$, the continuum space of paths. Often, these paths are not parametrized directly by their geometry but instead are described by their means of generation. For instance, some planners consider only straight-line paths. Given a current robot state $x_0 \in X$, the configuration space, a straight-line path is uniquely specified by connecting x_0 to an arbitrary sampled state $x_f \in X$. In such planners, it is expected that the robot is able to execute arbitrary paths, and so the boundary value problem is easy to solve because it is under-constrained.

Definition 7 (boundary value problem) Given start and end states, the **boundary value problem** (BVP) is to find any feasible path from the start to the goal (i.e. the local planning problem). A variant of this problem is to find the shortest such path. □

Some classes of robotic systems possess velocity constraints that limit the direction in which they may move instantaneously. The most well known example of these nonholonomic constraints is the difficulty of parallel parking a car. In such highly constrained, underactuated systems, the set of feasible paths $\mathcal{F}$ is much smaller than the space of all paths, $\mathcal{X}$. Thus, an arbitrary path sample drawn from $\mathcal{X}$ is unlikely to be in the feasible set $\mathcal{F}$. In such cases, the BVP is difficult to solve.

For constrained systems, we may avoid solving the BVP by instead sampling in U, the space of actions. Suppose we have a "black box" model of the robot's response to a control input, which is a mapping $M : U \to \mathcal{F}$. Sampling in the control space offers several advantages:

1. all sampled paths trivially obey motion constraints; and
2. we may precompute a set of paths and properties of those paths.

For a mobile robot, these paths are independent of initial position and heading, depending only on their derivatives (we ignore external interference such as wind or wheel slip). Therefore, a relatively small lookup table suffices to describe an expressive set of robot motions.

6.2 Prior Work

The motion planning community has invested considerable effort in the topic of non-uniform and adaptive sampling. The literature on this topic largely concerns probabilistic roadmaps (PRMs), which sample states rather than paths. Hsu et al. [62] provide a survey of recent work in non-uniform sampling for PRMs. We touch on a few of the broad approaches here.

One approach employs a fixed strategy to bias configuration sampling towards narrow corridors—parts of the C-space that are less likely to be sampled on their own due to their small measure—including papers by Amato et al. [3], Hsu et al. [60], V. Boor [135]. These works all restrict the sampling consideration to points, whereas we sample directly in the space of paths, using a distribution that varies in reaction to new collision test results.

The non-uniform sampling field has largely moved towards such adaptive strategies. For instance, Jaillet et al. [65] restrict sampling to size-varying balls around nodes in an RRT to avoid testing paths that would go through obstacles. Yu and Gupta [149] perform sensor-based planning in which a PRM is incrementally constructed based on the robot's partial observations of obstacles. Exploratory motions are selected by maximizing information gain. Another recent adaptive approach is to construct a meta-planner with several tools at its disposal; such planners employ multiple sampling strategies, as with Hsu et al. [61] or multiple randomized roadmap planners, as with Morales et al. [102], based on a prediction of which approach is most effective in a given setting.

One important feature of our work is the use of information from all collision tests, including positive results, to minimize entropy in a model approximating obstacle locations. The work of Burns and Brock [20, 21, 22, 23] bears considerable resemblance to ours in this regard. They describe an adaptive model of obstacle locations in C-space based on previous collision test results. Their model utilizes locally weighted learning to select state [21] or path [22] samples that reduce model uncertainty. We likewise develop a model of obstacle location, although ours inhabits the workspace and is simpler and more suitable for realtime applications. Burns and Brock subsequently observe, as we do, that model refinement is not an end in itself, but merely a means to the end of finding collision-free paths [23]. We proceed from this observation to consider what level of refinement is appropriate, in the context of constrained paths, based on the maximum width of corridor we are willing to miss discovering.

Separately, Burns and Brock describe an entropy-guided approach to the selection of configuration samples likely to unify two connected components of the PRM graph [20]. In later work, Burns and Brock [23] augment this approach with the notion of utility, which combines information gain regarding obstacle locations with the value of a resulting sample for solving the planning problem. Utility-guided sampling selects the configuration expected to solve the eventual planning problem most efficiently. We take a similar approach, in that we sample a combination of paths intended to navigate the space and to refine our obstacle model.

6.3 Informed Path Sampling Approach

In closing the loop on path sampling, we must feed back knowledge of obstacles reachable by the robot (in the form of collision-test results) into the sample space of paths, be it X or U, so

that we can suppress from the sampled path sequence future paths intersecting those regions of the workspace. Obstacles reside in the workspace, W, which may be represented as $\mathbb{R}^2$ or $\mathbb{R}^3$, depending on the nature of the robot. We may describe two set-valued functions,

$$ws\colon X \rightarrow \text{collection of subsets of } W$$
$$cs\colon W \rightarrow \text{collection of subsets of } X.$$

The function $ws(x)$ returns a set of workspace points $w_i \in W$ that the robot occupies while in configuration x, which is simply the Minkowski sum of the robot's shape with a point. $cs(w)$ returns the set of robot configurations $x_i \in X$ in which the robot intersects the workspace point w. Supposing that w resides inside an obstacle, these functions enable us to reason about configurations the robot must avoid.

For unconstrained systems that sample paths via points in the configuration space, closing the loop on path sampling requires finding the intersection of $cs(w_i)$ with the new point or path for each known collision point w_i. Motion planners do not typically attempt such elimination because the sampling process becomes exactly the collision test that it replaces. When the set of feasible motions is highly constrained, we prefer to sample paths by sampling in the space of actions. Computing the set of actions that result in the robot passing through a certain point in space is non-trivial since it requires solving a version of the BVP.

Instead, our approach to path sampling feedback in constrained systems is to keep the feedback entirely within the action space. We first collision-test some paths drawn from a low-dispersion sequence as given by Green and Kelly [48]. After finding the first collision point, its location biases future action samples. We may construct, a priori, a list of correspondences between action samples to accelerate this process.

A collision can be described as an ordered pair $c = (p, s)$, with $p \in \mathcal{F}$ and $s \in I = [0, 1]$, a time/distance parameter describing where on the path the collision occurred. A path is a mapping $p : I \rightarrow X$. Thus, c maps directly into a state $x \in X$, identifying the location of an obstacle. However, this collision state is special because it is known to be reachable by an action $u \in U$. In fact, x is probably reachable by a continuum of other actions, of which we can easily precompute a sampled subset for each possible collision point.

Often, collision-test routines are able to identify the precise location where a collision occurred. Knowing that point $w \in ws(x)$ is part of an obstacle, we may eliminate from our sampled sequence all paths passing through the set $cs(w)$. To eliminate these paths, we must store the list of actions by which they are parametrized.

In order to identify the set of paths passing unacceptably close to an obstacle point w, we precompute a proximity look-up table (PLUT), as shown in Fig. 6.2. Suppose our precomputed path set $\mathcal{P}$ contains N paths, each discretized into M points. The PLUT stores, for every ordered pair of paths (p_i, p_j) in $\mathcal{P}$, the shortest distance to the kth discretized point on p_j:

$$\text{PLUT}(p_i, p_j, k) = \min_{w \in p_i} d\left(w, p_j\left(\frac{k}{M}\right)\right), \tag{6.1}$$

where $d(w_1, w_2)$ gives the Euclidean distance between two points.

Now, given a collision $c = (p_j, s)$, we would like to find out if another path p_i would collide

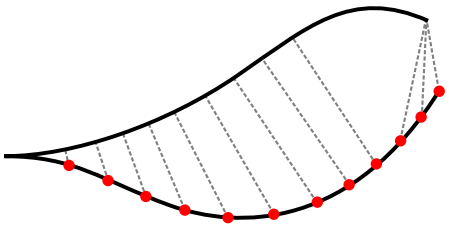


Figure 6.2 Given a path set of N paths, each discretized into M points, the proximity look-up table (PLUT) stores for each ordered pair of paths a list of shortest distances to each discrete point on the second path. Thus, there are a total of MN^2 unique PLUT entries.

with the same obstacle. We simply query the PLUT as

$$\mathrm{PLUT}(p_i, p_j, sM) < r_r, \tag{6.2}$$

where r_r is the radius of the robot (or an inscribed circle of the robot). When this condition holds, the collision test would fail. Knowing this, we may eliminate the path without a test, and spend the CPU time considering other paths.

However, we may go beyond short-circuiting the collision-test to estimating the probability distribution on obstacle locations using the *principle of locality*, which states that points inside an obstacle tend to occur near other points inside an obstacle. We propose a series of models of locality and two path sampling problems, which we address using our models. The key to success of this approach is that the final evaluation be less costly than the collision tests it replaces.

6.4 Probabilistic Foundations

We develop a series of probabilistic models that enable us to rapidly select paths for collision test that maximize one of two properties. First, in order to find valid robot motions, we must sample a selection of collision-free paths for execution. Second, we wish to sample broadly within the free space of paths, including in proximity to obstacles. The precision with which we know the obstacle/free-space boundaries directly relates to the size of narrow corridor we expect to find.

The workspace comprises a set of points divided into two categories: obstacle and free. The function

$$\mathrm{obs}\colon W \to \beta, \tag{6.3}$$

where $\beta = \{\text{true}, \text{false}\}$, reveals the outcome that a particular workspace point w is either inside (true) or outside of an obstacle. Building on such outcomes, we then describe an event of interest. A path collision-test takes the form

$$\mathrm{pct}\colon \mathcal{P} \to \beta, \tag{6.4}$$

which returns the disjunction of $\mathrm{obs}(w)$ for all w within the swath (Def. 3) of the path. A result of true indicates that this path intersects an obstacle.

Using these concepts as a basis, we pose two problems:

1. **Exploitation**. We are given a set of workspace points inside obstacles, $C = \{w_1, \ldots w_m\}$ and a set of untested paths $\mathcal{P}_{unknown} = \{p_1, \ldots p_n\}$. Knowing only a finite subset of the continuum of obstacle-points, find the path that minimizes the probability of collision:

$$p_{next} = \underset{p_i \in \mathcal{P}}{\operatorname{argmin}} \Pr(\operatorname{pct}(p_i, C)). \tag{6.5}$$

2. **Exploration**. Suppose we have a model of uncertainty $U(\mathcal{P}_{safe}, C)$ over the collision status of a set of untested paths, $\mathcal{P}_{unknown}$, in terms of a set of tested paths and known obstacle-points. Find the path $p_{next} \in \mathcal{P}_{unknown}$ giving the greatest reduction in expected uncertainty:

$$\begin{aligned} U_{exp}(p_i) = U\left(\mathcal{P}_{safe} \cup \{p_i\}, C\right) \Pr\left(\neg \operatorname{pct}(p_i, C)\right) \\ + U\left(\mathcal{P}_{safe}, C \cup \{c_i\}\right) \Pr\left(\operatorname{pct}(p_i, C)\right) \end{aligned} \tag{6.6}$$

$$p_{next} = \underset{p_i \in \mathcal{P}_{unknown}}{\operatorname{argmax}} \; U(\mathcal{P}_{safe}, C) - U_{exp}(p_i). \tag{6.7}$$

These two considerations are essentially the same as those encapsulated in the utility function of Burns and Brock [23] (see Section 6.2).

6.5 Locality

Thus far, we have demonstrated how a single failed collision test may serve to eliminate an entire set of untested paths from consideration because they pass through the same obstacle point. We may extend this approach one step further using the principle of locality.

Definition 8 (principle of locality) The **principle of locality** states that if a robot state is in collision with an obstacle, then that state is contained in a neighborhood ball of obstacle points.□

Two contributing factors combine to produce the locality effect. First, the non-zero volume of the robot means that even a point obstacle results in a set of robot states $cs(t)$ in collision with that point. The second factor is that real-world obstacles occupy some volume in space.

Given a known collision point, we employ the principle of locality to define a function expressing the probability that a new path under test is in collision with the same obstacle. A locality model takes the following general form:

$$\operatorname{loc}(p_i \mid C) = \Pr(\operatorname{pct}(p_i) \mid C) \tag{6.8}$$

Here, C may be a single point collision outcome or a set of collisions. If omitted, it is assumed to be the set of all known collisions.

This function depends on many factors, including the size and shape of the robot as well as the distribution on size and shape of obstacles in the environment. The most important parameter, however, is the distance between the new path and the known collision site. Thus, we may establish a rapidly computable first-order locality model in which we abstract away the size and shape of obstacles using a single distribution on radius, as in Fig. 6.3. We next discuss several intermediate locality model formulations before coming to the final form.

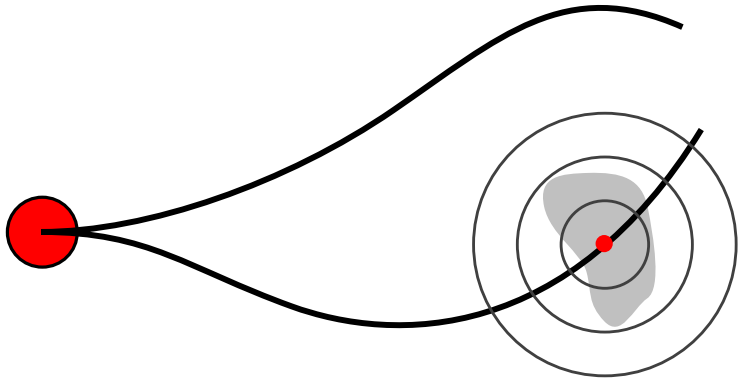


Figure 6.3 The robot (red disc at left) considers two paths. The bottom path fails its collision test. The locality model does not know the full extent of the obstacle (gray), but it can approximate the obstacle using a probability distribution (concentric circles) and can estimate the likelihood of the top path colliding.

6.5.1 General Locality Model

By explicitly modeling locality, we may reason about which paths are more or less likely to be in collision with any known obstacle, even with only partial information about its location. A path sampling algorithm, when informed by a locality model, provides a path sequence ordered by likelihood of collision, given currently known collision sites. We propose here a general model of locality that can be expected to produce collision-free path samples with high probability.

In constructing a general locality model, we abstract away many parameters; we consider both the robot and obstacles to be balls (in $\mathbb{R}^2$ or $\mathbb{R}^3$), and the obstacles are assumed uniform in radius. We relax some of these assumptions later, in Section 6.5.4. For now, these restrictions permit us to simplify the model by removing bearing from consideration. Thus, the general model's prediction of future collisions is purely a function of *range* from the known collision site to the path. The fixed radii of both the obstacles (r_o) and the robot (r_r) result in the intuitive notion of locality: that its influence is over a limited range only.

The precise formulation of the general locality model, as depicted in Fig. 6.4, is based on maintaining a probability distribution on possible locations of the obstacle centroid, given a known collision site. In this naive model, the location of the centroid is described by a uniform distribution over $B(r_o)$, a ball of radius r_o centered at the colliding position of the robot. A path p_i sweeps out a swath $S(p_i)$ of width $2r_o + 2r_r$. Any non-empty set $B(r_o) \cap S(p_i)$ represents some probability of collision. This general model then predicts that the probability of collision is

$$\mathrm{loc}_{general}(p_i \mid c) = \frac{|B(r_o) \cap S(p_i)|}{|B(r_o)|}. \tag{6.9}$$

If we regard p_i as a straight line, then in 2D the probability of collision is the ratio of the area of a circular segment to the area of the whole circle, which is [11]:

$$f_{segment}(r) = \begin{cases} \frac{1}{\pi r_e^2}\left(r_e^2 \cos^{-1}\frac{r-r_e}{r_e} - (r-r_e)\sqrt{2r_e r - r^2}\right) & \text{if } 0 \le r \le 2r_e \\ 0 & \text{otherwise,} \end{cases} \tag{6.10}$$

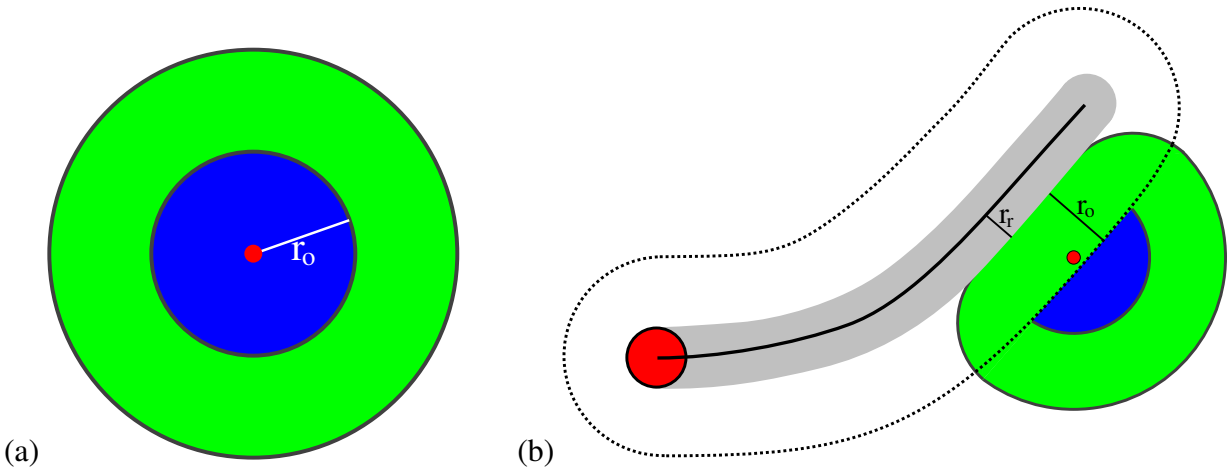


Figure 6.4 The general locality model: **(a)** Given a point known to be in collision with an obstacle (center red dot), the blue inner circle of radius r_o represents possible locations of the centroid of the obstacle. The green outer circle comprises points possibly occupied by some part of the obstacle. **(b)** The probability that a new candidate robot path is collision-free equals the fraction of possible obstacle centroids outside a swath of width $2r_o + 2r_r$. The blue region represents the set of possible centroids, while the green region depicts possible obstacle extents.

where r is the range between the path and the collision point. We call r_e the range of effect, which we set equal to r_o here.

Definition 9 (range of effect) The **range of effect** of a known collision point describes the radius around that point at which paths are regarded to be at risk of collision with the known obstacle. □

6.5.2 Simple Locality Model

We now propose an even simpler locality model, which closely approximates (6.10) but makes use of the existing PLUT. For paths bounded in curvature, the above model may be closely approximated by considering only the point on the new path most closely approaching the known collision point. This new locality model employs the raised cosine distribution, due to its finite support and resemblance to (6.10)—as seen in Fig. 6.5. The raised cosine distribution is expressed as

$$f_{rcd}(r) = \begin{cases} \frac{1}{2r_e}\left[1 + \cos\left(\pi \frac{r}{2r_e}\right)\right] & \text{if } 0 \leq r \leq 2r_e \\ 0 & \text{otherwise.} \end{cases} \tag{6.11}$$

Then, the probability that a new path p_i will collide with the same obstacle represented by the previous collision $c = (p_j, s)$ is simply

$$\text{loc}_{simple}(p_i \mid c) = f_{rcd}(\text{PLUT}(p_i, p_j, sM) - r_r). \tag{6.12}$$

Note that here we are no longer maintaining an explicit probability distribution on the location of an obstacle but instead a heuristic estimate of the risk of a single path relative to a single collision site.

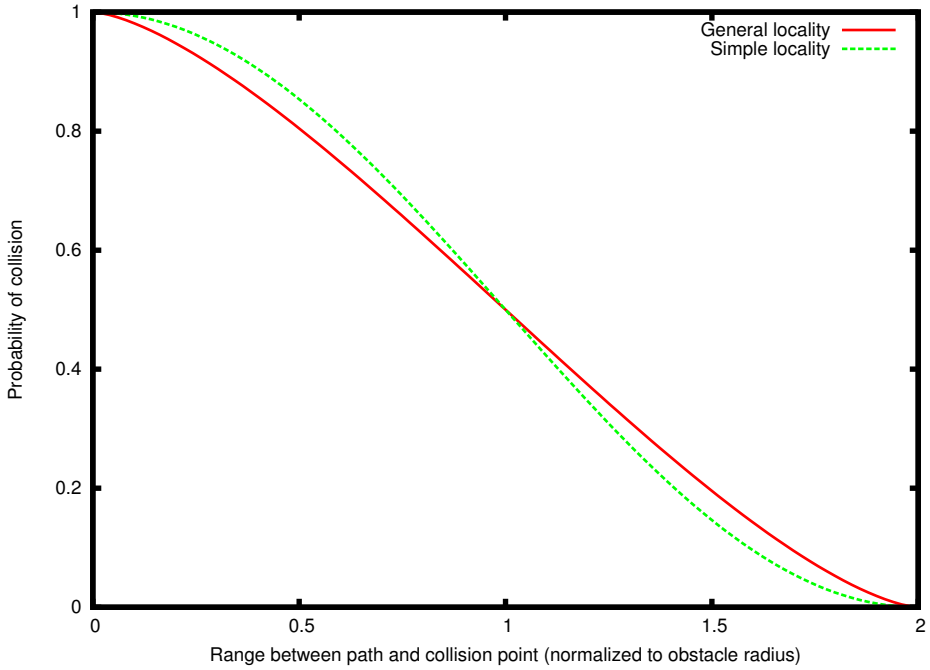


Figure 6.5 Comparison of General Locality Model and Simple Locality Model. The latter employs the raised cosine distribution.

6.5.3 Handling Multiple Collision Sites

Given a known collision site, both (6.9) and (6.12) provide a tool for selecting a candidate path to minimize the probability of collision. However, we have not yet addressed the issue of multiple known collision sites. The likelihood that two workspace points have the same obstacle outcome correlates strongly with the distance between them, by virtue of describing the same obstacle. The estimate of collision likelihood for an untested path depends on what statistical independence assumptions we make among known collision points.

Fig. 6.6 depicts a situation in which two collision sites appear to be correlated. However, many possible policies for estimating statistical independence among a set of collision points, such as clustering, are complex to compute and implement.

In contrast, we may conservatively assume that all collisions are independent, in which case basic probability theory states that

$$\mathrm{loc}(p) = \mathrm{loc}(p \mid \{c_1, \ldots, c_n\}) = 1 - \prod_{i \in \{1,\ldots,n\}} (1 - \mathrm{loc}(p \mid c_i)). \tag{6.13}$$

If some collision sites are actually part of the same obstacle, then we are overestimating the likelihood of collision for p. In the absence of any knowledge regarding correlation, however, the most conservative policy is the safest. Thus, we now have the means to address Problem 1, Exploitation:

$$p_{next} = \operatorname*{argmin}_{p_i \in \mathcal{P}} \mathrm{loc}(p_i). \tag{6.14}$$

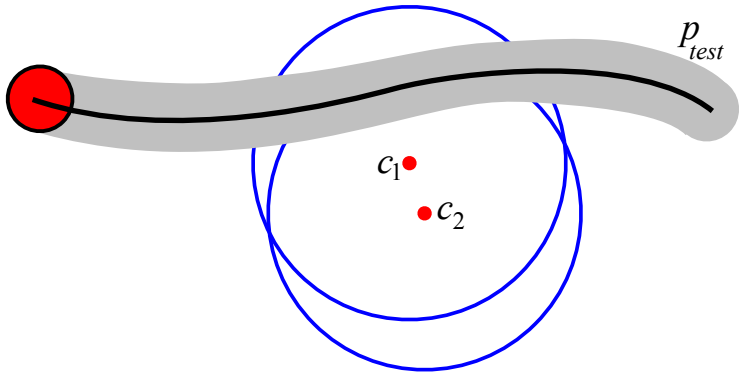


Figure 6.6 Two collision sites c_1 and c_2 are located in close proximity. Intuition suggests that c_2 should be ignored when computing the risk of collision of path p_{test}. Either the two sites belong to the same obstacle, or else the obstacle at c_2 is "blocked" by the obstacle at c_1.

In the next section, we explore an information theoretic approach to safely adjusting this pessimistic model.

6.5.4 Adaptive Locality Model

The locality models presented in Sections 6.5.1–6.5.2 incorporate only positive collision test outcomes. Those static models conservatively estimate an obstacle distribution spread over a large but finite range of effect. We now construct an *adaptive* locality model capable of incorporating both positive and negative collision-test outcomes.

If we should happen to discover a safe path p_{safe} passing within collision site c's range of effect, then we may use this new information to refine the obstacle model of c. In effect, we adjust the locality function to act over a smaller range in the direction of p_{safe}. As Fig. 6.7a shows, no path p_{test} that is separated from c by p_{safe} can possibly be at risk of collision with this obstacle. This adaptive model effectively relaxes the earlier, rigid assumptions on obstacle size and independence of collision sites. In modeling such geometric relations, we depart from prior work addressing locality.

Following an update to the model, all future probability estimates involving c incorporate this new information. Although the independence assumption may initially make nearby paths like p_{test} appear riskier than they should (Fig. 6.6), the adaptive model rapidly cancels out this effect after finding a safe path to shrink each collision point's range of effect.

In addressing the problem of how to adaptively adjust obstacle distributions in reaction to a collision-free path, a variety of approaches present themselves. One possible approach is to shrink the range of effect for the obstacle at c, as in Fig 6.7b, which supposes that the obstacle is smaller than initially thought. Another approach, to shift the entire distribution away from the safe path as in Fig. 6.7c, assumes that the obstacle size was correctly estimated, but its position was off.

We adopt a compromise position. We prefer that the collision site remains the center of a distribution in order to keep range checks efficient via look-up table. However, we also prefer to avoid altering the range of effect of the opposing side, about which we have no new data.

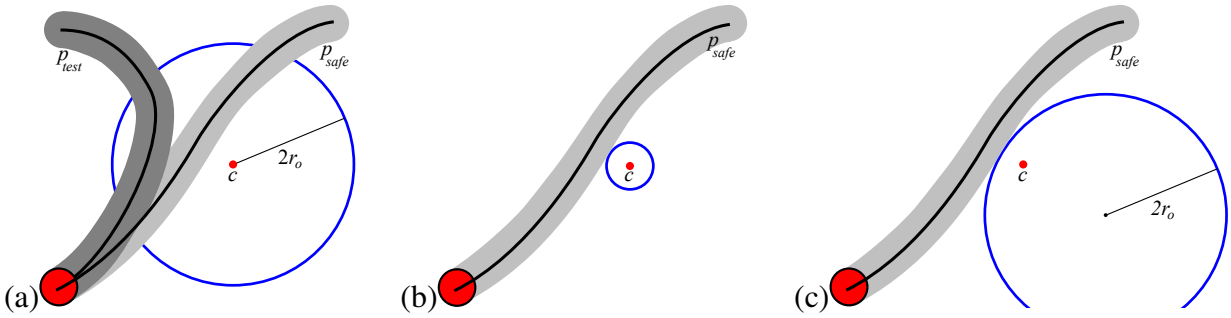


Figure 6.7 (a) Given a collision point c and neighboring collision-free path p_{safe}, the blue circle represents a distribution on obstacle locations, some of which are invalidated by p_{safe}. The more distant candidate path p_{test} is not at risk of collision with the obstacle represented by c.
(b) and **(c)** Two simple hypotheses on obstacle scale and position explain these two results. The distribution shown in Fig. 6.4(b) is simpler to represent during online path sampling.

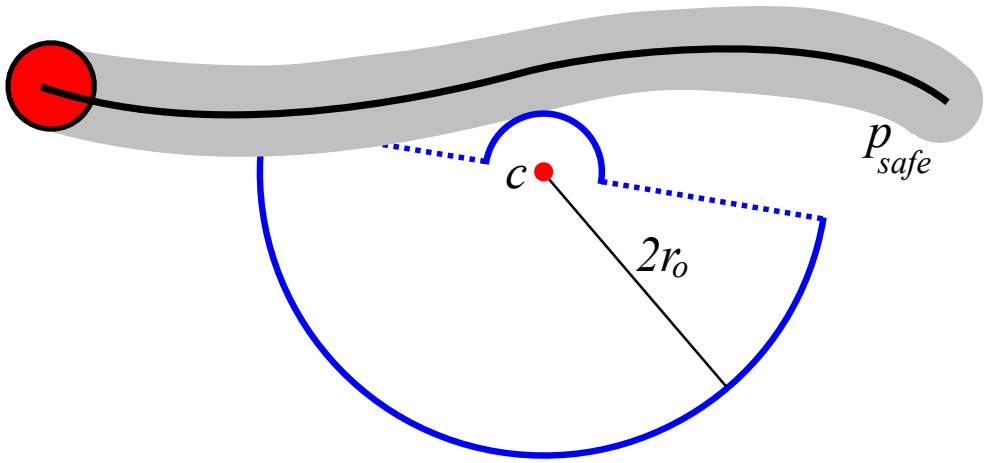


Figure 6.8 The range of effect on each side of collision site c is maintained separately. The left range began at $2r_o$, but it was reduced after successfully collision-testing path p_{safe}.

We therefore split the range of effect into several regions of influence ("sides") centered around each collision site. In 2D, we have left and right sides of the obstacle, as in Fig. 6.8. In 3D, the division is topologically more arbitrary, although we split the obstacle into four sides.

In splitting the locality model into several directions, we require a rule to consistently associate each path with a particular side of the collision point. The sides are defined relative to the pose of the robot before executing the path. The sides meet at the line $\mathbf{a}$, an axis running through the start pose and the collision point. We assign names to the sides describing their position relative to the robot's frame of reference. Sides are determined by

$$\textit{left} = \text{sgn}(\mathbf{t} \times \mathbf{p} \cdot \mathbf{u}) \tag{6.15}$$

$$\textit{top} = \text{sgn}(\mathbf{t} \times \mathbf{p} \cdot \mathbf{a} \times \mathbf{u}), \tag{6.16}$$

where $\mathbf{u}$ denotes the robot's up vector, $\mathbf{p}$ the projection of c onto the path, and $\mathbf{t}$ the tangent vector of the path at this point, as in Fig. 6.9. These sides may be precomputed for each path.

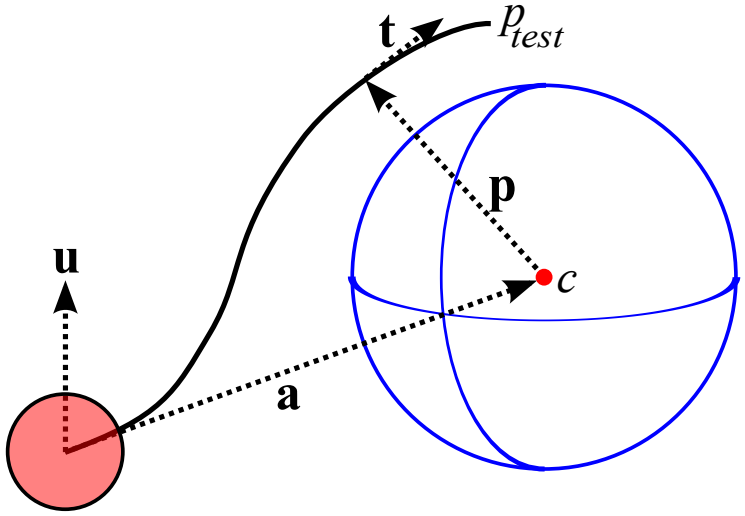


Figure 6.9 In three dimensions, the adaptive locality model's range of effect is split into four sides. The robot's up vector, **u**, and the vector pointing toward the collision point, **a**, are used to define which of four sides the path p_{test} is on. As illustrated, the path is on the top-left side.

In 2D, it is particularly convenient to augment the PLUT with a sign indicating on which side of the path each possible collision point lies.

Fig. 6.10 shows a family of paths on the left side of an obstacle. We deem each path equally likely to collide with the obstacle because they each approach equally near to the collision point, c. This assignment of paths to a single side of an obstacle places assumptions on the path's shape. We assume here that curvature is bounded and that paths are reasonably short. See Chapter 7 for a thorough discussion of these assumptions.

6.6 Path Entropy

Having established an adaptive locality model, we now consider a means to reap maximal advantage from its predictive capabilities in order to solve Problem 2, Exploration. It is important to select paths for collision test that cause the model to rapidly converge to an accurate description of obstacles, while simultaneously minimizing failed collision tests. Given a set of collision sites, the best path to collision test is that path with maximum entropy according to the current model parameters.

Definition 10 (path entropy) An untested path's **entropy** (sometimes called Shannon entropy) refers to the expected amount of information about the safety of other untested paths that would be gained from collision-testing it. A path's entropy is

$$H(\mathrm{pct}(p)) = -\Pr(\mathrm{pct}(p)) \log \Pr(\mathrm{pct}(p)) - \Pr(\neg\,\mathrm{pct}(p)) \log \Pr(\neg\,\mathrm{pct}(p)). \tag{6.17}$$

□

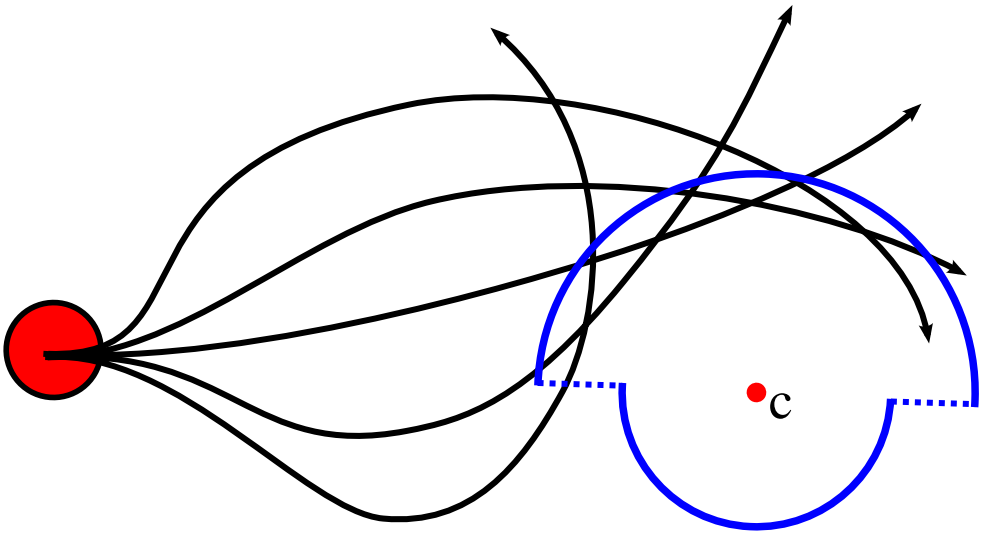


Figure 6.10 A family of paths, all of which pass to the left of the collision site, c. Despite the variety of shapes, each path intrudes equally into the left range of effect of c, and thus they would each reduce its left range of effect equally.

In order to maximize our understanding of the true distribution of obstacles with the fewest possible samples, we choose to sample the maximum entropy path:

$$p_{test} = \underset{p_i \in \mathcal{P}}{\operatorname{argmax}}\ H(\mathrm{pct}(p_i)). \tag{6.18}$$

Based on current information, the maximum entropy path has maximal uncertainty with regard to its collision with obstacles; its probability of collision is nearest to 50%. Testing this particular path will therefore increase total knowledge more than any other. The result will be either a path that significantly reduces the range of effect for some known collision point(s) or a new collision point that is far from known collisions. In either case, model accuracy increases with maximal utility.

The policy of maximizing entropy was proposed by Jaynes [67] for the purpose of estimating an unknown distribution. Maximum entropy has been specifically applied to decision theory, as by Grünwald and Dawid [49], and as we employ it here. The decision that maximizes entropy is the one that minimizes the worst case possible outcome. In our case, the worst outcome is rediscovering a known result because it wastes computation time for no gain. This outcome takes two forms:

1. testing a path that passes through (or very near) a known collision site, or
2. retesting a known safe path, or one very similar in shape.

By maximizing the worst-case outcome, this Γ-minimax approach, described by Vidakovic [139], is capable of reasoning simultaneously about an entire family of probability distributions, called Γ—in our case, a range of theories about obstacle extent.

If the maximum entropy policy is pursued repeatedly, path selection proceeds to discover a sequence of safe paths and collision sites that are progressively nearer to each other, thus

establishing precisely the boundaries separating the obstacles from free space. Knowing these boundaries may accelerate the process of sampling and testing paths more densely within the free space. In Chapter 7, we provide one possible approach to this process.

Refining these boundaries is a process of diminishing returns, however. As we discover safe paths progressively closer to obstacles, the margin of uncertainty becomes so low that additional maximum entropy path samples provide negligible advantage for a variety of reasons:

1. the cost of collision testing a path often increases with greater obstacle proximity;
2. there is a computational cost associated with path sampling that is proportional to the number of known collision sites; and
3. paths close to obstacles are poor choices for execution.

These factors could be incorporated into a utility function, as Burns and Brock [23] did, but this remains as future work for us. Thus, we should not exclusively pursue the maximum entropy sampling policy, but also select path samples far from obstacles to maximize safety and path diversity.

We utilize several strategies to combine exploration and exploitation in a hybrid approach. In the absence of any uncertainty from our locality model (such as before the first collision site has been discovered), we sample from a low-dispersion sequence. In the presence of uncertainty, we compute the fraction f of the total replan cycle time that has already elapsed. With probability f we pursue an exploitation (obstacle avoidance) sampling strategy, whereas with probability $1 - f$ we instead pursue an exploration (boundary finding) strategy to refine our locality model. Sampled paths very near to known obstacles are set aside without testing for later use, since they make poor candidates for traversal. We search these paths if there is time at the end of the replan cycle, after testing all other paths.

6.7 Experimental Results

We conducted a set of experiments in simulation in order to obtain a sufficient quantity of trials to recognize statistically meaningful trends. Experimental setup is as described in Section 3.2.3, in which trials comprise sets of one-hundred planning problems; in each one, the curvature-constrained robot attempts to navigate through randomly generated 2D environments, each based on a query comprising start and goal poses. The robot moves continuously while replanning a local path. A heuristic function selects goal-directed paths. In order to assess the effect of various path sampling strategies, we varied two parameters: obstacle density, and replan cycle time. Fig. 6.11 shows a screen capture of our simulator, indicating known collision sites (in C-space) and ranges of effect. We consider four path sampling strategies:

- Low dispersion—sequence generated by Green and Kelly [48]
- Avoid obstacles—pure exploitation; sample as far as possible from obstacles
- Find boundaries—pure exploration; selects maximum entropy path
- Hybrid approach—combines “avoid obstacles” and “find boundaries” strategies.

We present results on three of the strategies in Fig. 6.12. We see an increase in safe paths produced per unit time for both pure obstacle avoidance (up to 7.8x) and the hybrid approach (up

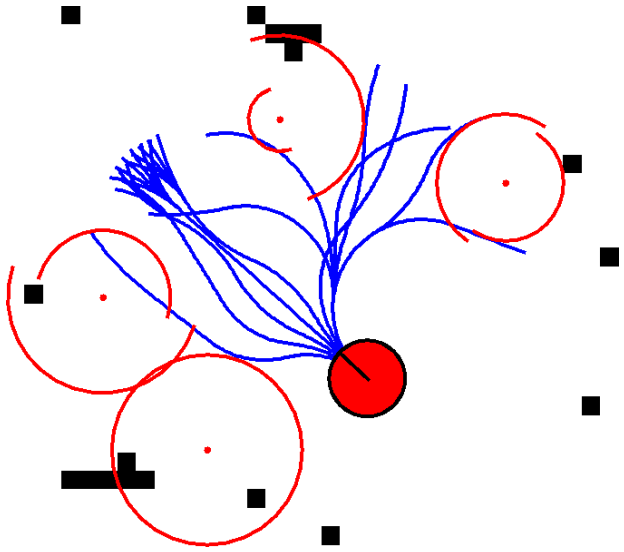


Figure 6.11 Simulator depiction of locality model, showing: collision-free paths (blue), known collision sites (red dots), and their ranges of effect (concentric semicircles). Note that the dots correspond to the nearest edge of each C-space obstacle—the point most relevant from the robot's current pose. Not all obstacles (black) are relevant to the current local plan.

to 3x), compared to a fixed low dispersion sequence. However, pure obstacle avoidance sampling suffers a drop in performance at solving planning queries, which the hybrid approach overcomes. Note that the low-dispersion sampler actually declines slightly in performance as more samples are allowed, which is consistent with the non-monotonicity property earlier results. The hybrid planner shows a trend of increasing performance as it improves its locality model.

6.8 Summary

In real time planning, performance is sensitive to the computational cost associated with collision testing. Alternatives that alleviate some of that computation can be beneficial, provided that such alternatives are computationally efficient themselves. In this chapter, we present a strategy for informed path sampling that guides the search away from obstacles and towards safe or unexplored parts of the workspace. Although obstacle information is already available to the planner in costmap form, we obtain a significant increase in performance by representing the most salient subset of those obstacles in a more immediately accessible form.

We utilize a proximity look-up table to accelerate this process. Even so, our statistical model describing nearby obstacles and their relationship is necessarily simple. This model makes use of the principle of locality to search appropriately far from obstacle locations already discovered in prior collision tests to maximally reduce uncertainty. Using our probabilistic locality model, we trade off between exploration and exploitation in order to discover a variety of safe paths while largely avoiding searching colliding paths.

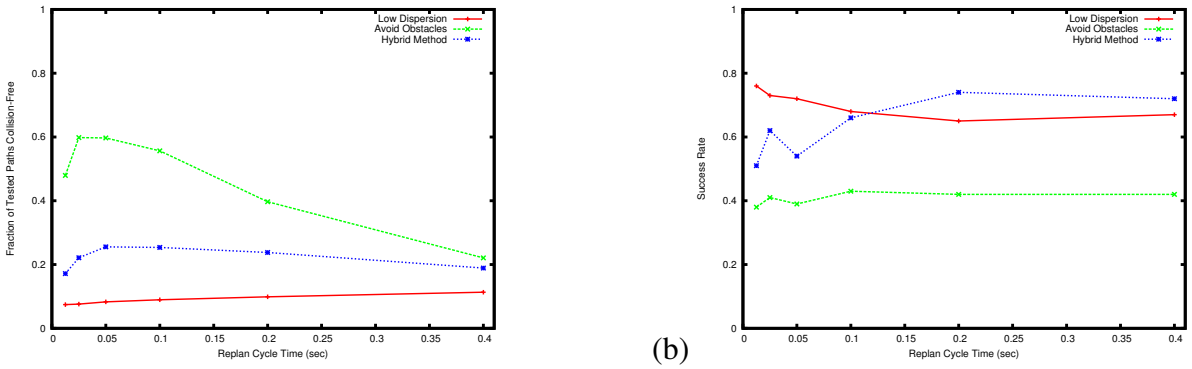


Figure 6.12 All tests in these plots were run at an obstacle density of 2%. **(a)** The locality model provides up to a 7.8x increase in the fraction of paths collision-free per replan cycle. As replan cycle time increases, we see regression toward the mean, as a larger total fraction of available paths are collision tested. After 0.4 secs, the low-dispersion approach has tested on average 90% of all paths, the "avoid obstacles" strategy has tested 45%, whereas the hybrid method has tested only 8% of available paths. **(b)** In success rate at solving planning queries, we see that the "avoid obstacles" strategy suffers in performance, whereas the hybrid approach, which strikes a balance between exploration and exploitation of the locality model, performs increasingly well as it has more time to gather information.

Chapter 7

Collision Testing

The bottleneck in path testing is collision checking [125]. In this chapter, we introduce a novel approach that delivers a significant increase in path set collision-testing performance by exploiting the fundamental geometric and topological structure of paths. This work was first presented by Knepper et al. [81].

We introduce an equivalence relation intuitively resembling the topological notion of homotopy. Two paths are *path homotopic* if a continuous, collision-free deformation with fixed start and end points exists between them [103]. Like any path equivalence relation, homotopy partitions paths into equivalence classes. Different homotopy classes make fundamentally different choices about their route amongst obstacles. However, two constraints imposed by mobile robots translate poorly into homotopy theory: limited sensing and constrained action.

The robot may lack a complete workspace map, which it must instead construct incrementally from sensor data. Since robot perception is limited by range and occlusion, a robot's understanding of obstacles blocking its movement evolves as it moves. A variety of sensor-based planning algorithms have been developed to handle such partial information. Obstacle avoidance methods, such as the potential fields of Khatib [78], the vector field histograms of Borenstein and Koren [15], and the curvature-velocity method of Simmons [129], are purely reactive. The bug algorithm by Lumelsky and Stepanov [99], which generates a path to the goal using only a contact sensor, is complete in two dimensional spaces. A planner using the hierarchical generalized Voronoi graph, a roadmap with global line-of-sight accessibility proposed by Choset and Burdick [29] achieves completeness in higher dimensions using range readings of the environment. Yu and Gupta [149] propose a planner that iteratively constructs a probabilistic roadmap in response to partial sensed information about the world. Actions are selected on the basis of maximizing information gain for future plan steps. Our local planner resembles these algorithms in that it reacts to local obstacles while receiving global guidance about the direction to the goal.

If a robot is tasked to perform long-range navigation, then it must plan a path through initially unsensed regions. In a hierarchical planning context, a low-fidelity global planner (i.e. one ignoring constraints) generates this path because we prefer to avoid significant investment in this plan, which will likely be invalidated later. Path homotopy, in the strictest sense, requires global knowledge of obstacles because homotopy equivalent paths must connect fixed start and goal points.

Relaxing the endpoint requirement of homotopy avoids reasoning about the existence of far-

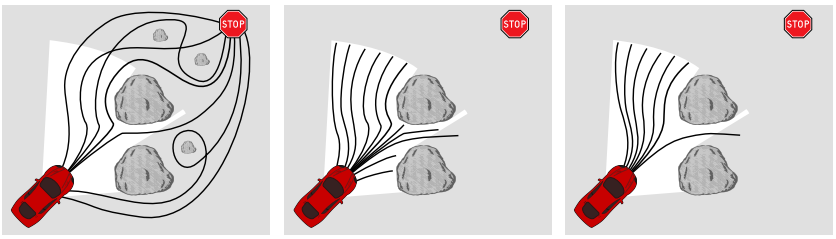


Figure 7.1 Motivation for a local equivalence relation. **left:** Paths from a few distinct homotopy classes between the robot and the goal. The distinctions between some classes require information that the robot has not yet sensed (the gray area is out of range or occluded). **middle:** With paths restricted to the sensed area, they may freely deform around visible obstacles. **right:** After restricting path shape to conform to motion constraints, we get a handful of equivalence classes that are immediately applicable to the robot.

away, unsensed obstacles. In naively relaxing a fixed endpoint, our paths might be permitted to freely deform around obstacles, making all paths equivalent (see Fig. 7.1). To restore meaningful equivalence classes, we propose an alternate constraint based on path shape. Such path shape constraints stem from the nonholonomic motion constraints inherent to many mobile robots. Laumond [91] first highlighted the importance of nonholonomic constraints and showed that feasible paths exist for a mobile robot with such constraints. Barraquand and Latombe [6] created a grid-based planner that innately captures these constraints. LaValle and Kuffner [94] proposed the first planner to incorporate both kinodynamic constraints and random sampling. In contrast to nonholonomic constraints, true homotopy forbids restrictions on path shape; two paths are equivalent if *any* path deformation exists between them. By restricting our paths to bounded curvature (or similarly, bounded controls), we represent only feasible motions while limiting paths' ability to deform around obstacles. The resulting set of path equivalence classes is of immediate importance to the planner (Fig. 7.1). The number of choices represented by these local equivalence classes relates to Farber's topological complexity of motion planning [37].

Various planners have employed equivalence classes to reduce the size of the search space. In task planning, recent work by Gardiol and Kaelbling [44] has shown that equivalence classes of actions can be used to eliminate redundant search. In motion planning, path equivalence often employs homotopy. A recent paper by Bhattacharya et al. [12] provides a technique based on complex analysis for detecting homotopic equivalence among paths in 2D. Two papers employing equivalence classes to build probabilistic roadmaps [72] are by Schmitzberger et al. [127] and Jaillet and Siméon [64]. The latter paper proposes the visibility deformation, a departure from true homotopic equivalence that restricts continuous deformation to line-of-sight visibility between paths. We propose here a different variation on homotopy. Not only do we restrict continuous deformation between paths, but we also fix path length to create a purely local path equivalence relation.

Our key insight in this chapter is that this *local path equivalence* reveals shared outcomes in collision-testing. Specifically, two equivalent neighboring paths represent swept volumes of the robot that cover some common ground in the workspace. Between them lies a continuum of paths whose swept volumes are covered by the first two. We develop the mathematical foundations to detect equivalence relations among all local paths based on a finite precomputed path set. We

then utilize these tools to devise efficient algorithms for detecting equivalence and **implicitly** collision-testing local paths (circumventing the normal, expensive test).

7.1 Algorithms

In this section, we present two new algorithms for path classification and implicit path collision-testing. We also borrow a path set generation routine from prior work.

7.1.1 Path Set Generation

We use the greedy path set construction technique of Green and Kelly [48], outlined in Alg. 3. Recall that the algorithm iteratively builds a path set sequence $\{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ by drawing paths from a densely-sampled source path set, $\mathcal{P}_N \subset \mathcal{X}$, the continuum path space. At step i, it selects the path $p \in \mathcal{P}_N$ that minimizes the dispersion (Def. 4) of $\mathcal{P}_i = \mathcal{P}_{i-1} \cup \{p\}$.

The Green-Kelly algorithm generates a sequence of path sets $\mathcal{P}_i$, for $i \in \{1, \ldots, N\}$, that has monotonically decreasing dispersion. In seeking a path to execute, the local planner algorithm (Alg. 1) searches paths in this order, thus permitting early termination while ensuring that a low-dispersion set of paths is collision tested. Note that although the source set $\mathcal{P}_N$ is of finite size—providing a lower bound on dispersion at runtime—it can be chosen with arbitrarily low dispersion in $\mathcal{X}$ a priori.

7.1.2 Path Classification

We present Alg. 4, which classifies collision-free members of a path set. Recall the Hausdorff metric, which is central to the algorithm. Intuitively, this metric returns the largest amount of separation between two paths in the workspace. From Munkres [103]:

$$\mu_H(p_i, p_j) = \inf_{\varepsilon} \{p_i \subset (p_j)_\varepsilon \text{ and } p_j \subset (p_i)_\varepsilon\}, \tag{7.1}$$

where $(p)_r$ denotes dilation of p by r: $\{t \in \mathbb{R}^2 \colon \|t_p - t\|_{L2} \leq r \text{ for some } t_p \in p\}$. Note that μ_H satisfies all properties of a metric [53]. For our fixed path set generated by Green-Kelly and a given d, we precompute each pairwise path metric value of (7.1) and store them in a lookup table for rapid online access.

Alg. 4 performs path classification on a set of paths that have already tested collision-free at runtime. We form an *equivalence graph* $G = (V, E)$ in which node $v_i \in V$ corresponds to path p_i. Edge $e_{ij} \in E$ exists, joining nodes v_i and v_j, when this relation holds:

$$\mu_H(p_i, p_j) \leq 1, \tag{7.2}$$

where d is the diameter of the robot. This condition is true when two paths are separated by at most one robot diameter. Taking the transitive closure of this relation, two paths p_a and p_b are equivalent if nodes v_a and v_b are in the same connected component of G (Fig. 7.2).

In effect, this algorithm constructs a probabilistic roadmap (PRM) in the path space instead of the conventional configuration space. A query into this PRM tells whether two paths are

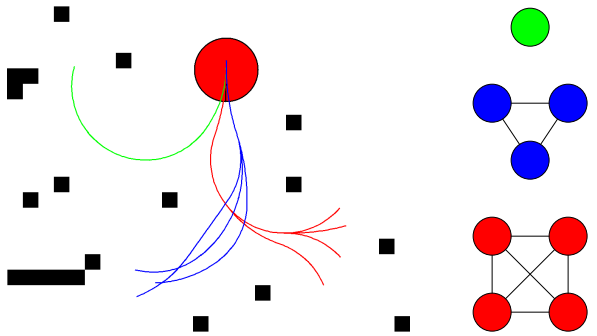


Figure 7.2 Computing path equivalence. A simple path set, in which obstacles (black) eliminate colliding paths. The collision-free path set has three equivalence classes (red, green, and blue). In the corresponding graph representation, at right, adjacent nodes represent proximal paths. Connected components indicate equivalence classes of paths.

equivalent. As with any PRM, a query is performed by adding two new graph nodes v_s and v_g corresponding to the two paths. We attempt to join these nodes to other nodes in the graph based on (7.2). The existence of a path connecting v_s to v_g indicates path equivalence.

7.1.3 Implicit Path Safety Test

There is an incessant need in motion planning to accelerate collision-testing, which may take up to 99% of total CPU time [125]. During collision-testing, the planner must verify that a given swath is free of obstacles.

Definition 11 (safe) We say a path is **safe** if its swath contains no obstacles. □

In testing many swaths of a robot passing through space, most planners effectively test the free workspace many times by testing overlapping swaths. We may test a path **implicitly** at significant computational savings by recalling recent collision-testing outcomes and circumventing new collision tests whenever possible. We formalize the idea in Alg. 5, which is designed to be invoked from Alg. 2, line 4 in lieu of the standard path test routine.

The implicit collision-test condition resembles the neighbor condition (7.2) used by Alg. 4, but it has an additional "Is_Between" check, which indicates that the swath of the path under test is fully covered by two collision-free neighboring swaths. The betweenness trait can be precomputed and stored in a lookup table. Given a set of safe paths, we can quickly discover whether any pair covers the path under test. Experimental results show that this algorithm allows us to test up to 90% of paths implicitly, thus increasing the path evaluation rate by up to 300% in experiments.

7.2 Foundations

In this section, we establish the foundations of an equivalence relation on path space based on continuous deformations between paths. We then provide correctness proofs for our algorithms for classification and implicit collision-testing.

Algorithm 4 $D \leftarrow$Equivalence_Classes($\mathcal{P}_{free}$, d)

Input: $\mathcal{P}_{free}$ – a set of safe, appropriate paths; d – the diameter of the robot
Output: D – a partition of $\mathcal{P}_{free}$ into equivalence classes (a set of path sets)

Let $G = (V, E) \leftarrow (\emptyset, \emptyset)$
$D \leftarrow \emptyset$ // Partition of paths into classes (represented by a set of sets)
for all $p_i \in \mathcal{P}_{free}$ **do** // This loop discovers adjacency
  $V.add(p_i)$ // Add a graph node corresponding to path p_i
  for all $p_j \in V \setminus \{p_i\}$ **do**
    if $\mu_H(p_i, p_j) \leq 1$ **then**
      $E.add(i, j)$ // Connect nodes i and j with an unweighted edge
    end if
  end for
end for
$\mathcal{S} \leftarrow \mathcal{P}_{free}$ // Unclassified paths
while $\mathcal{S} \neq \emptyset$ **do** // This loop finds the connected components
  $\mathcal{C} \leftarrow \emptyset$ // Next connected component
  $p \leftarrow$ a member of $\mathcal{S}$
  $\mathcal{L} \leftarrow \{p\}$ // List of nodes to be expanded in this class
  while $\mathcal{L} \neq \emptyset$ **do**
    $p \leftarrow$ a member of $\mathcal{L}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$ // Commit p to class
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{p\}$
    $\mathcal{L} \leftarrow (\mathcal{L} \cup V.neighbors(p)) \cap \mathcal{S}$
  end while
  $D \leftarrow D \cup \{\mathcal{C}\}$
end while
return D

We are given a kinematic description of paths. All paths are parametrized by a common initial pose, common fixed length, and individual curvature function. Let $\kappa_i(s)$ describe the curvature of path i as a function of arc length, with $\max_{0 \leq s \leq s_f} |\kappa_i(s)| \leq \kappa_{max}$. Typical expressions for κ_i include polynomials, piecewise constant functions, and piecewise linear functions. The robot motion produced by control i is a *feasible* path given by

$$\begin{bmatrix} \dot{\theta}_i(t) \\ \dot{x}_i(t) \\ \dot{y}_i(t) \end{bmatrix} = \begin{bmatrix} \kappa_i(t) \\ \cos\theta_i(t) \\ \sin\theta_i(t) \end{bmatrix}. \tag{7.3}$$

Definition 12 (feasible) A **feasible** path has bounded curvature (implying at least C^1 continuity) and fixed length. The set $\mathcal{F}(s_f, \kappa_{max})$ contains all feasible paths of length s_f and curvature $|\kappa(s)| \leq \kappa_{max}$. □

Algorithm 5 $b \leftarrow$Test_Path_Implicit(p, w, $\mathcal{S}$, d)

Input: p is a path to be tested
Input: w is a costmap object // used as a backup when path cannot be implicitly tested
Input: $\mathcal{S}$ is the set of safe paths found so far
Input: d is the diameter of the robot
Output: b – boolean indicating whether path is safe

```
for all p_i, p_j ∈ S such that μ_H(p_i, p_j) ≤ 1 do
    if p.Is_Between(p_i, p_j) then                    // p's swath has been tested previously
        s_f ← p.Get_End_Point()
        collision ← w.Test_Point(s_f)                 // endpoint may not be covered by swaths
        return collision
    end if
end for
return w.Test_Path(p)                                 // Fall back to explicit path test
```

7.2.1 Properties of Paths

In this section, we establish a small set of conditions under which we can quickly determine that two paths are equivalent. We constrain path shape through two dimensionless ratios relating three physical parameters. We may then detect equivalence through a simple test on pairs of paths using the Hausdorff metric.

These constraints ensure a continuous deformation between neighboring paths while permitting a range of useful actions. Many important classes of action sets obey these general constraints, including the line segments common in RRT [94] and PRM [72] planners, as well as constant curvature arcs. Fig. 5.1 illustrates some more expressive action sets that adhere to our constraints.

The three related physical parameters are: d, the diameter of the robot; s_f, the length of each path; and r_{min}, the minimum radius of curvature allowed for any path. Note that $1/r_{min} = \kappa_{max}$, the upper bound on curvature, is a constant. For non-circular robots, d reflects the minimal cross-section of the robot's swath sweeping along a path. We express relationships among the three physical quantities by two dimensionless parameters:

$$v = \frac{d}{r_{min}} \qquad\qquad w = \frac{s_f}{2\pi r_{min}}.$$

We only compare paths with like values of v and w. Fig. 7.3 provides some intuition on the effect of these parameters on path shape. Due to the geometry of paths, only certain choices of v and w are appropriate.

Definition 13 (appropriate) An **appropriate path** is a feasible path conforming to appropriate values of v and w from the proof of Lemma 2. Fig. 7.3 previews the permissible values. □

When the condition in (7.2) is met, the two paths' swaths overlap, resulting in a continuum of coverage between the paths. This coverage, in turn, ensures the existence of a continuous deformation, as we show in Theorem 1, but first we formally define a continuous deformation between paths.

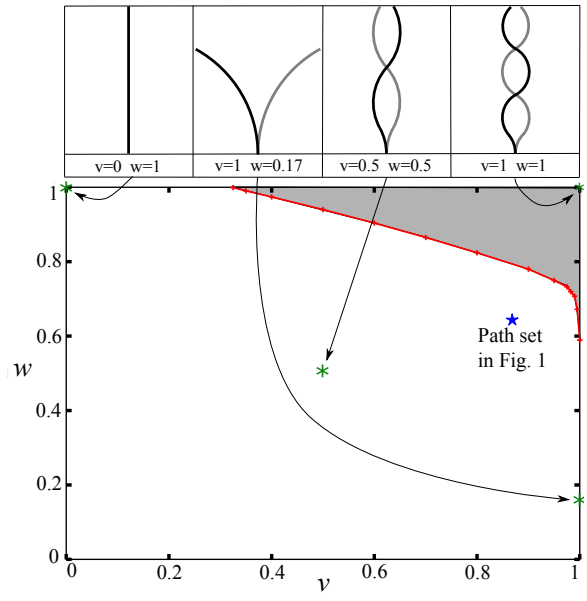


Figure 7.3 Appropriate paths. **At top:** several example paths combining different values of v and w. Each path pair obeys (7.2). The value of v affects the "curviness" allowed in paths, whereas w affects their length.
At bottom: this plot, generated numerically, approximates the set of appropriate choices for v and w. The gray region at top right must be avoided, as we show in Lemma 2. Such choices would permit an obstacle to occur between two safe paths that obey (7.2). A path whose values fall in the white region is called an *appropriate path*.

Definition 14 (continuous deformation) A **continuous deformation** between two safe, feasible paths p_i and p_j in $\mathcal{F}(s_f, \kappa_{max})$ is a continuous function $f\colon [0,1] \rightarrow \mathcal{F}(s_f^-, \kappa_{max}^+)$, with s_f^- slightly less than s_f and κ_{max}^+ slightly more than κ_{max}. $f(0)$ is the initial interval of p_i, and $f(1)$ is the initial interval of p_j, both of length s_f^-. □

The length s_f^- depends on v and w, but for typical values, s_f^- is fully 95–98% of s_f. For many applications, this is sufficient, but an application can quickly test the remaining path length if necessary. Nearly all paths $f(c)$ are bounded by curvature κ_{max}, but it turns out that in certain geometric circumstances, the maximum curvature through a continuous deformation is up to $\kappa_{max}^+ = \frac{4}{3}\kappa_{max}$. This limit occurs in the event that a point on the medial axis has two nearest neighbor points sharing curvature κ_{max} in the same direction.

Definition 15 (equivalent) We write $p_i \sim p_j$ to indicate that a continuous deformation exists between paths p_i and p_j, and they are therefore **equivalent**. □

Definition 16 (guard paths) Two safe, feasible paths that define a continuous deformation are called **guard paths** because they protect the intermediate paths. □

In the presence of obstacles, it is not trivial to determine whether a continuous deformation is safe, thus maintaining equivalency. Rather than trying to find a deformation between arbitrary paths, we propose a particular condition under which we show that a bounded-curvature, fixed-

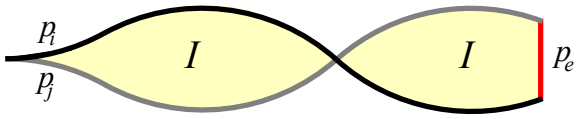


Figure 7.4]
Paths p_i, p_j, and p_e form boundary B. Its interior, I, contains all paths in the continuous deformation from p_i to p_j. The set of paths in I illustrates the betweenness trait described in Section 7.1.3.

length, continuous path deformation exists,

$$\mu_H(p_1, p_2) \leq 1 \implies p_1 \sim p_2. \tag{7.4}$$

This statement, which we prove in the next section, is the basis for Alg. 4 and Alg. 5. The overlapping swaths of appropriate paths p_1 and p_2 cover a continuum of intermediate swaths between the two paths. The equivalence relation, of which (7.4) detects local instances, is a proper equivalence relation because it possesses each of three properties:

- **reflexivity**. $\mu_H(p, p) = 0$; p is trivially deformable to itself.
- **symmetry**. The Hausdorff metric is symmetric.
- **transitivity**. Given $\mu_H(p_1, p_2) \leq 1$ and $\mu_H(p_2, p_3) \leq 1$, a continuous deformation can be constructed from p_1 to p_3 passing through p_2.

7.2.2 Equivalence Relation

We now prove (7.4); that is, we show that shape constraints indicated by v and w combined with Hausdorff distance constraints are sufficient to ensure the existence of a continuous deformation between two neighboring paths. Our approach to the proof will be to first describe a feasible continuous deformation, then show that paths along this deformation are safe.

Given appropriate guard paths p_i and p_j with common origin, let p_e be the shortest curve in the workspace connecting their endpoints without crossing either path (p_e may pass through obstacles). The closed path $B = p_i + p_e + p_j$ creates one or more closed loops (the paths may cross each other). In the case of a two-dimensional workspace, we employ the Jordan curve theorem [103], which tells us that each loop partitions $\mathbb{R}^2$ into two sets, only one of which is compact. Let I, the interior, be the union of these compact sets with B, as in Fig. 7.4.

Definition 17 (between) A path p_c is **between** paths p_i and p_j if $p_c \subset I$. □

In the case of a three-dimensional workspace, it becomes necessary to select a 2D manifold containing both paths, such that each path $f(c)$ within the continuous deformation is entirely contained within the manifold. In general, more than one such viable manifold exists. We defer the selection of a specific manifold to later in this section. First, we show the existence of a feasible continuous deformation between guard paths under specified conditions in two dimensions.

Lemma 1 *Given appropriate paths $p_i, p_j \subset \mathcal{F}(s_f, \kappa_{max})$ with $\mu_H(p_i, p_j) \leq 1$, a path sequence exists in the form of a feasible continuous deformation between p_i and p_j.* □

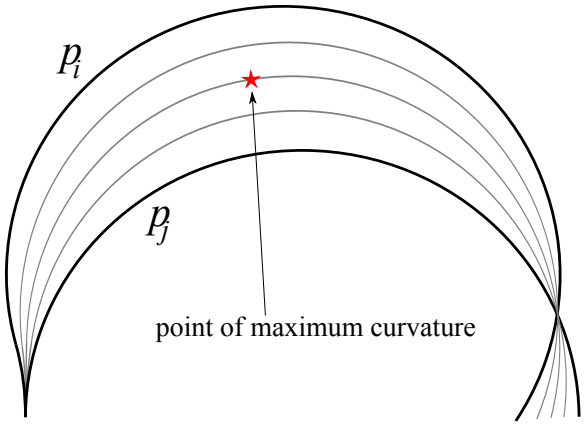


Figure 7.5 In a continuous deformation between paths p_i and p_j, as defined by the level sets of (7.6), each path takes the form of a weighted GVD. Upper bounds on curvature vary along the deformation, with the maximum bound of $\frac{4}{3}\kappa_{max}$ occurring at the medial axis of the two paths.

PROOF We provide the form of a continuous deformation from p_i to p_j such that each intermediate path is between them. With t a workspace point and p a path, let

$$\gamma(t,p) = \inf_{\varepsilon} t \in (p)_\varepsilon \tag{7.5}$$

$$g(t) = \begin{cases} [0,1] & \text{if } \gamma(t,p_i) = \gamma(t,p_j) = 0 \\ \left\{ \frac{\gamma(t,p_i)}{\gamma(t,p_i)+\gamma(t,p_j)} \right\} & \text{otherwise,} \end{cases} \tag{7.6}$$

where $g(t)$ is a set-valued function to accommodate intersecting paths. Each level set $g(t) = c$ for $c \in [0,1]$ defines a weighted generalized Voronoi diagram (GVD) forming a path as in Fig. 7.5. We give the form of a continuous deformation using level sets $g^{-1}(c)$; each path is parametrized starting at the origin and extending for a length s_f^- in the direction of p_e.

Let us now pin down the value of s_f^-, the length of intermediate paths p_c. Every point t_i on p_i forms a line segment projecting it to its nearest neighbor t_j on p_j (and vice versa). Their collective area is shown in Fig. 7.6. Eqn. (7.2) bounds each segment's length at d. s_f^- is the greatest value such that no intermediate path of length s_f^- departs from the region covered by these projections.

For general-shaped generators in $\mathbb{R}^2$, the GVD forms a set of curves meeting at branching points [123]. In this case, no GVD cusps or branching points occur in any intermediate path. Since $d < r_{min}$, no center of curvature along either guard path can fall in I [13]. Therefore, each level set defines a unique path through the origin.

Each path's curvature function is piecewise continuous and everywhere bounded. A small neighborhood of either guard path approximates constant curvature. A GVD curve generated by two constant-curvature sets forms a conic section [148]. Table 7.1 reflects that the curvature of p_c is everywhere bounded with the maximum possible curvature being bounded by $\frac{4}{3}\kappa_{max}$. For the full proofs, see [80]. Thus, each intermediate path p_c is a feasible path. ■

In order to extend our concept of continuous deformations to three dimensional workspaces,

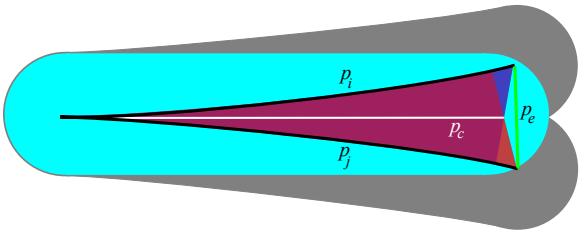


Figure 7.6 Hausdorff coverage (overlapping red and blue shapes in center) is a conservative approximation of swath coverage (gray). The Hausdorff distance between paths p_i and p_j is equal to the maximum-length projection from any point on either path to the closest point on the opposite path. Each projection implies a line segment. The set of projections from the top line (blue) and bottom line (red) each cover a solid region between the paths. These areas, in turn, cover a slightly shorter intermediate path p_c, in white, with its swath in cyan. This path's length, s_f^- is as great as possible while remaining safe, with its swath inside the gray area.

Table 7.1 Conic sections form the weighted Voronoi diagram. κ_1 and κ_2 represent the curvatures of the two guard paths, with κ_1 the lesser magnitude. Let $\kappa_m = \max(|\kappa_1|, |\kappa_2|)$. For details, see [80].

Type	Occurrence	Curvature bounds of intermediate paths
line	$\kappa_1 = -\kappa_2$	$\lvert\kappa\rvert \leq \kappa_m$
parabola	$\kappa_1 = 0, \kappa_2 \neq 0$	$\lvert\kappa\rvert \leq \kappa_m$
hyperbola	$\kappa_1\kappa_2 < 0, \kappa_1 \neq -\kappa_2$	$\lvert\kappa\rvert \leq \kappa_m$
ellipse	$\kappa_1\kappa_2 > 0$	$\lvert\kappa\rvert < \frac{4}{3}\kappa_m$

we recognize that the set of paths comprising the weighted GVD forms a 2D manifold containing both guard paths:

$$M = \bigcup_{c\in[0,1]} g(c). \tag{7.7}$$

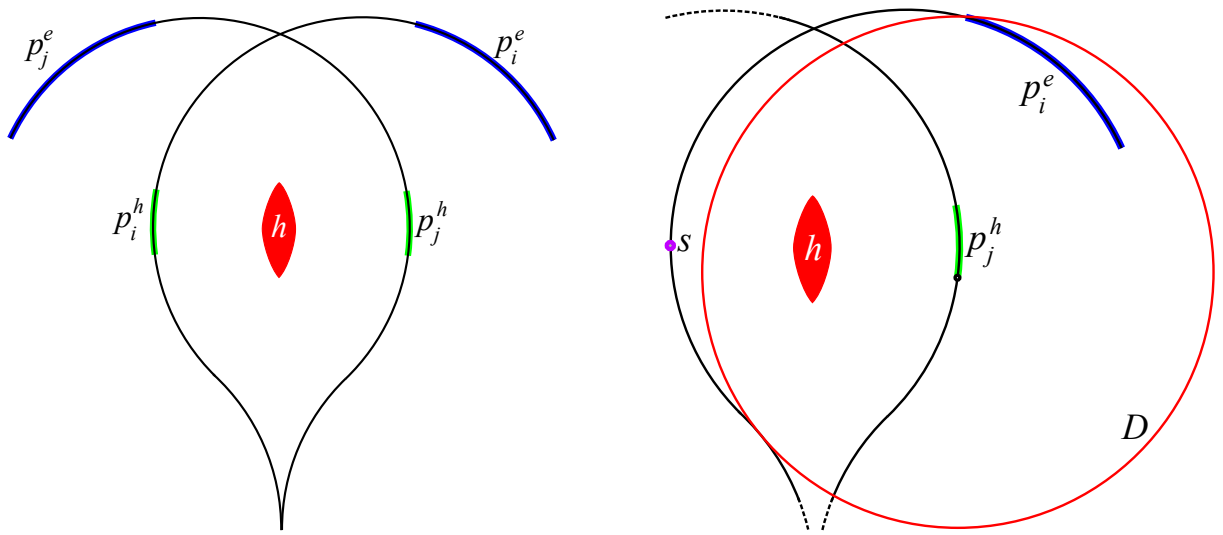
This manifold is not the only legitimate container for a continuous deformation between paths. The full discussion of continuous deformations in 3D is left as future work.

Lemma 2 *Given safe, appropriate guard paths $p_i, p_j \in \mathcal{F}(s_f, \kappa_{max})$ separated by $\mu_H(p_i, p_j) \leq 1$, any path $p_c \subset \mathcal{F}(s_f^-, \frac{4}{3}\kappa_{max})$ between them is safe.* □

PROOF We prove this lemma by contradiction. Assume an obstacle lies between p_i and p_j. We show that this assumption imposes lower bounds on v and w. We then conclude that for lesser values of v and w, no such obstacle can exist.

Let $sl(p, d) = \{t \in \mathbb{R}^2, t_p = nn(t, p) \colon \overline{t_p t} \perp p \text{ and } \|t - t_p\|_{L2} \leq \frac{d}{2}\}$ define a conservative approximation of a swath, obtained by sweeping a line segment of length d with its center along the path. $\overline{t_p t}$ is the line segment joining t_p to t and $nn(t, p)$ is the nearest neighbor of point t on path p. The two swaths form a safe region, $U = sl(p_i, d) \cup sl(p_j, d)$.

Suppose that U contains a hole, denoted by the set h, which might contain an obstacle. Now, consider the shape of the paths that could produce such a hole. Beginning with equal position and heading, they must diverge widely enough to separate by more than d. To close the loop in U, the paths must then bend back towards each other. Since the paths separate by more than d,



(a) With bounded curvature, there is a lower bound on path lengths that permit a hole, h, while satisfying (7.2), indicated by p_i^e, the blue highlight. Shorter path lengths ensure the existence of a safe continuous deformation between paths.

(b) We compute the maximal path length that prevents a hole using Vendittelli's solution to the shortest path for a Dubins car. Starting from the dot marked s, we find the shortest path intersecting the circle D of radius r_{min}. The interval p_i^e illustrates path lengths permitting a hole to exist. Shorter paths leave some part of p_j^h uncovered.

Figure 7.7 Finding upper bound on path length.

there exist two open intervals $p_i^h \subset p_i$ and $p_j^h \subset p_j$ surrounding the hole on each path such that (at this point) $p_i^h \not\subset (p_j)_d$ and $p_j^h \not\subset (p_i)_d$. To satisfy (7.2), there must exist later intervals $p_i^e \subset p_i$ such that $p_j^h \subset (p_i^e)_d$ and likewise $p_j^e \subset p_j$ such that $p_i^h \subset (p_j^e)_d$, as in Fig. 7.7a.

How long must a path be to satisfy this condition? Consider the minimum length solution to this problem under bounded curvature. For each point $t \in p_j^h$, the interval p_i^e must intersect the open disc $D = \text{int}((t)_d)$, as in Fig. 7.7b. Since p_j^h grows with the width of h, and p_i^e must intersect all of these open neighborhoods D, the path becomes longer with larger holes. We will therefore consider the minimal small-hole case.

Vendittelli et al. [138] solve the shortest path problem for a Dubins car to reach a line segment. We may approximate the circular boundary of D by a set of arbitrarily small line segments. One may show from this work that given the position and slope of points along any such circle, the shortest path to reach its boundary (and thus its interior) is a constant-curvature arc of radius r_{min}. In the limit, as v approaches one and the size of h approaches zero, the length of arc needed to satisfy (7.2) approaches $\pi/2$ from above, resulting in the condition that $w > 0.48$. Thus, for $w \leq 0.48$ and $v \in [0, 1)$, p_c is safe. For smaller values of v, D shrinks relative to r_{min}, requiring longer paths to reach, thus allowing larger values of w as shown in the plot in Fig. 7.3.

We have shown that there exist appropriate choices for v and w such that (7.2) implies that U contains no holes. Since U contains the origin, any path $p_c \in I$ emanating from the origin passes through U and is safe. ■

Theorem 1 *Given safe, appropriate guard paths $p_i, p_j \in \mathcal{F}(s_f, \kappa_{max})$, and given $\mu_H(p_i, p_j) \leq$*

1, *a safe continuous deformation exists between* p_i *and* p_j. □

PROOF Lemma 1 shows that (7.6) gives a continuous deformation between paths p_i and p_j such that each intermediate path $p_c \subset I$ is feasible. Lemma 2 shows that any such path is safe. Therefore, a continuous deformation exists between p_i and p_j. This proves the validity of the Hausdorff metric as a test for path equivalence. ■

By chaining together continuous deformations between neighboring paths, we can demonstrate that a continuous deformation exists between any pair of paths within an equivalence class by following the correct sequence of edges of the equivalence graph. This property holds for any paths in our discretely sampled set. It also applies for any other pair of paths satisfying the shape constraints, provided that the discrete sampling is sufficiently dense. The existence of a sufficiently dense path sampling is the subject of the next section.

7.2.3 Resolution Completeness of Path Classifier

In this section, we show that Alg. 4 is resolution complete. Resolution completeness commonly shows that there exists a sufficiently high discretization of each dimension of the search space such that the planner finds a path exactly when one exists in the continuum space. We instead show that there exists a sufficiently low dispersion sampling in the infinite-dimensional path space such that the approximation given by Alg. 4 has the same connectivity as the continuum safe, feasible path space.

Let $\mathcal{F}$ be the continuum feasible path space and $\mathcal{F}_{free} \subset \mathcal{F}$ be the set of safe, feasible paths. Using the Green-Kelly algorithm, we sample offline from $\mathcal{F}$ a path sequence $\mathcal{P}_N$ of size N. At runtime, using Alg. 2, we test members of $\mathcal{P}_N$ in order to discover a set $\mathcal{P}_{free} \subset \mathcal{P}_N$ of safe paths.

The following lemma is based on the work of LaValle et al. [95], who prove resolution completeness of deterministic roadmap (DRM) planners, which are PRM planners that draw samples from a low-dispersion, deterministic source. Since we use a deterministic sequence provided by Green-Kelly, the combination of Alg. 2 and 4 generates a DRM in path space.

Lemma 3 *For any given configuration of obstacles and any path set* $\mathcal{P}_N$ *generated by the Green-Kelly algorithm, there exists a sufficiently large* N *such that any two paths* $p_i, p_j \in \mathcal{P}_{free}$ *are in the same connected component of* $\mathcal{F}_{free}$ *if and only if Alg. 4 reports that* $p_i \sim p_j$. □

PROOF LaValle et al. [95] show that by increasing N, a sufficiently low dispersion can be achieved to make a DRM complete in any given C-Space. By an identical argument, given a continuum connected component $\mathcal{C} \subset \mathcal{F}_{free}$, all sampled paths in $\mathcal{C} \cap \mathcal{P}_N$ are in a single partition of $\mathcal{P}_{free}$. If q is the radius of the narrowest corridor in $\mathcal{C}$, then for dispersion $\delta_N < q$, our discrete approximation exactly replicates the connectivity of the continuum freespace. ■

Lemma 4 *Under the same conditions as in Lemma 3, there exists a sufficiently large* N *such that for any continuum connected component* $\mathcal{C} \subset \mathcal{F}_{free}$*, Alg. 2 returns a* $\mathcal{P}_{free}$ *such that* $\mathcal{P}_{free} \cap \mathcal{C} \neq \emptyset$*. That is, every component in* $\mathcal{F}_{free}$ *has a corresponding partition returned by Alg. 4.* □

PROOF Let B_r be the largest open ball of radius r in $\mathcal{C}$. When $\delta_N < r$, B_r must contain some sample $p \in \mathcal{P}_N$. Since $\mathcal{C}$ is entirely collision-free, $p \in \mathcal{P}_{free}$. Thus, for dispersion less than r, $\mathcal{P}_{free}$ contains a path in $\mathcal{C}$. ■

There exists a sufficiently large N such that after N samples, $\mathcal{P}_N$ has achieved dispersion $\delta_N < \min(q, r)$, where q and r are the dispersion required by Lemmas 3 and 4, respectively. Under such conditions, a bijection exists between the connected components of $\mathcal{P}_{free}$ and $\mathcal{F}_{free}$.

Theorem 2 *Let $D = \{\mathcal{D}_1|\ldots|\mathcal{D}_m\}$ be a partition of $\mathcal{P}_{free}$ as defined by Alg. 4. Let $C = \{\mathcal{C}_1|\ldots|\mathcal{C}_m\}$ be a finite partition of the continuum safe, feasible path space into connected components. A bijection $f : D \rightarrow C$ exists such that $\mathcal{D}_i \subset f(\mathcal{D}_i)$.* □

PROOF Lemma 3 establishes that f is one-to-one, whereas Lemma 4 establishes that f is onto. Therefore, f is bijective. This shows that by sampling at sufficiently high density, we can achieve an arbitrarily good approximation of the connectedness of the continuum set of collision-free paths in any environment. ■

Finally, we move on to show that we can detect path safety while circumventing a collision test.

Theorem 3 *A path interval p_c may be implicitly tested safe if it is between paths p_i and p_j such that $\mu_H(p_i, p_j) \leq 1$ and a small region at the end of p_c has been explicitly tested.* □

PROOF By Lemma 2, the initial interval of p_c is safe because its swath is covered by the swaths of the guard paths. Since the small interval at the end of p_c has been explicitly tested, the whole of p_c is collision-free. ■

7.3 Experimental Results

We present some simulation results involving equivalence class detection and implicit path collision testing. All tests were performed in simulation on planning problems of the type described in Section 3.2.3. During navigation, the local planner is permitted to run for a fixed amount of time within each replan cycle before executing its chosen path. A variable number of paths will be tested each cycle depending on factors such as obstacle clutter and implicit path testing. In some experiments, we vary the planning time allotted, whereas other experiments explore the effects of obstacle density.

7.3.1 Classification Performance Overhead

Path classification imposes a computational overhead due to the cost of searching for neighboring collision-free paths. Collision rate in turn relates to the density of obstacles in the environment. Fig. 7.8 shows that the computational overhead of our classification implementation is nearly 20% in an empty environment but drops to 0.3% in dense clutter. It is in precisely such high-clutter environments that the usefulness of classification is maximized since two arbitrary paths are less likely to be equivalent amongst many obstacles. We now proceed to weigh these and other benefits of path classification against its costs.

7.3.2 Collision Testing

Regardless of obstacle density, implicit collision-testing more than compensates for the overhead of path classification. Fig. 7.9 shows the effect of implicit path testing on total paths tested in

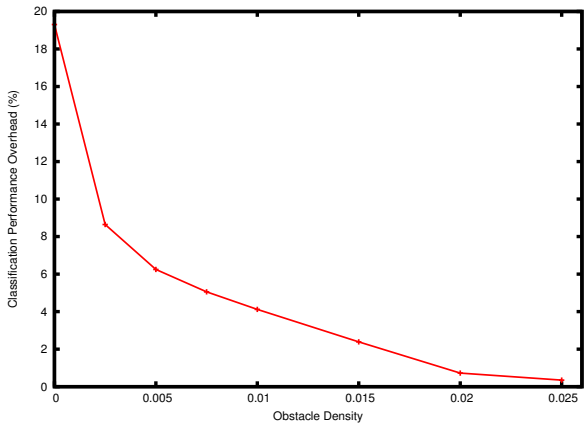


Figure 7.8 Path classification overhead is minimized in exactly those densely-cluttered problems where its contribution is most valuable.

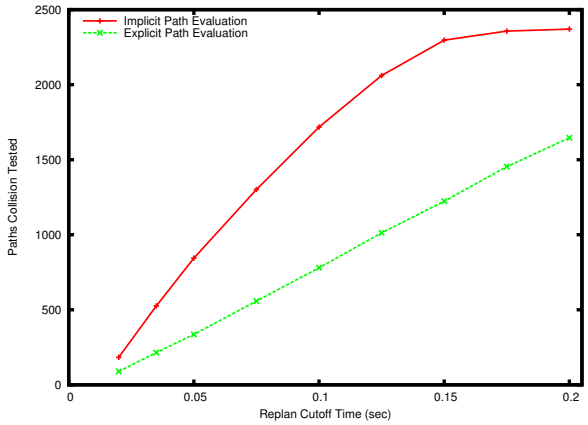


Figure 7.9 Paths tested per time-limited replan step in an obstacle-free environment. Increased replan time corresponds with a larger effective path set size and greater path density. Path testing performance improves by up to 3x with the algorithms we present here. Note that an artificial ceiling curtails performance at the high end due to a maximum path set of size 2,401.

the absence of obstacles. We compare the implicit collision-tester of Alg. 5 against traditional explicit collision-testing. As the time allotment for testing paths increases, the number of paths collision-tested under the traditional algorithm increases linearly at a rate of 8,300 paths per second. With implicit testing, the initial test rate over small time allotments (thus small path set sizes) is over 22,500 paths per second. The marginal rate declines over time due to the aforementioned overhead, but implicit path testing still maintains its speed advantage until the entire 2,401-member path set is collision-tested. Note that this result occurs in the empty world case, where overhead is most severe.

Fig. 7.10 presents implicit collision-testing performance in the presence of clutter. As obstacle density increases, we expect overhead to drop, but it simultaneously becomes more difficult to satisfy the necessary conditions for implicitly testing a path. Fixing the replan rate at an intermediate value of 10 Hz, we see that implicit path evaluation maintains an expected advantage

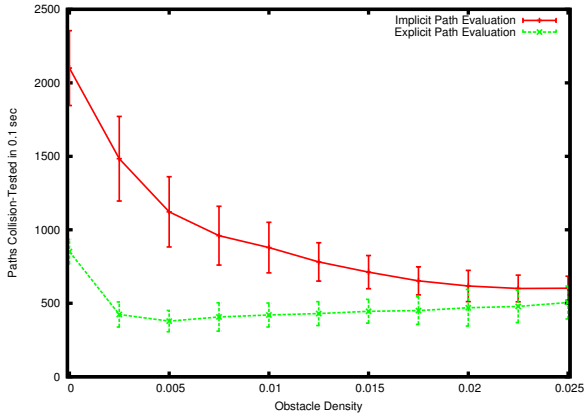


Figure 7.10 Paths tested per 0.1 second time step at varying obstacle densities. Implicit collision-testing allows significantly more paths to be tested per unit time. Even in extremely dense clutter, implicit path testing considers an extra 19% of paths on average. The right edge of the graph represents the maximum density at which environments remain navigable. Error bars indicate 95% confidence.

across all navigable obstacle densities. In high clutter, this advantage is statistically less clear, yet implicit path testing still outperforms explicit path testing with over 90% confidence at the maximum tested obstacle density.

Chapter 8

Path Selection

A mobile robot navigating through the world faces two categories of choices. Where corridors split from each other, there is a discrete decision problem. A robot approaching a decision point at constant velocity has a limited time in which to freely select a corridor without the penalty of backtracking. Such decisions trade off among length, complexity, and safety of routes to the goal.

A different sort of choice is the continuous optimization involved in selecting a path while traveling within a free space, such as in a corridor. In this case, the robot typically trades off risk of collision with shortness of path. We illustrate these two types of choice in Fig. 8.1.

These two qualitatively distinct choices are reflected in the variety of approaches to motion planning. Many planners decompose the continuum search space into a graph, thus transforming the planning problem into a graph search (a discrete decision process). Early examples deterministically decompose space into a regular grid, including Barraquand and Latombe [6], who discretize the configuration space and action space to generate graph nodes and edges. Other deterministic decision-based methods construct a graph adaptively based on the geometric shape of obstacles, such as the work of Choset and Burdick [29]. In more recent years, probabilistic planners have become popular. Algorithms like probabilistic roadmaps (PRMs) [72], rapidly exploring random trees [93], and lazy PRMs [124] take advantage of the asymptotic low dispersion of random sequences to sample uniformly throughout complex and high-dimensional configuration spaces without prescribing a fixed density a priori.

Another set of algorithms directly considers motion in the continuum. Khatib's potential fields [78] produce a smooth path in the C-Space by following the gradient of a function combining weighted penalty terms for obstacle proximity with a reward term for progress toward the goal. Through tuning the weights on these terms, a robot may achieve reasonable goal-directed behavior, but the formulation is subject to local minima in of the potential field so that the robot may fail to reach its goal. Rimon and Koditschek proposed the navigation function [121], which assures a potential function free of local minima under certain geometric assumptions on obstacles. Ratliff, et al. introduce CHOMP [115], which performs functional gradient descent on a C-Space path to improve on an initial naive straight-line path by minimizing depth of penetration into obstacles.

All of the above motion planning approaches view the problem as either purely discrete or purely continuous. By contrast, there has been a limited amount of work on hybrid planners that

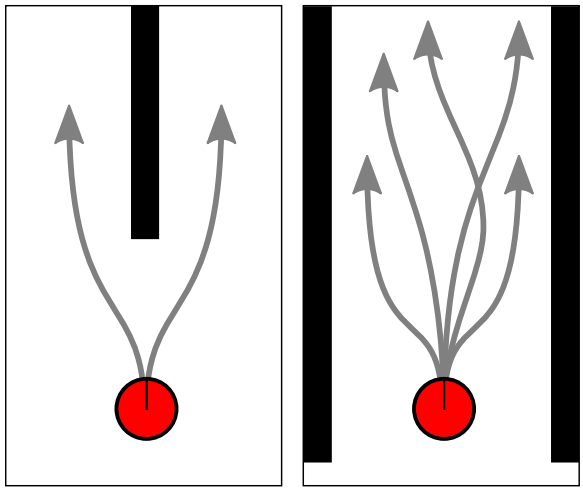


Figure 8.1 A navigating robot faces both discrete decisions (left) about which corridor to follow and continuous optimization (right) over where in the corridor to drive. A planner or controller should be able to consider these choices separately.

reason about both decision problems and optimization problems. The elastic bands of Quinlan and Khatib [113] optimize an initial collision-free C-Space path generated by a discretizing grid planner—using a functional gradient technique similar to CHOMP—to keep the path safely away from obstacles, even if the obstacles move during execution. The elastic strips of Brock and Khatib [18] operate similarly but in the robot's workspace, thereby alleviating the problem of high-dimensional search for highly articulated systems. Since elastic bands, elastic strips, and CHOMP all operate by decoupling global planning and local optimization, they each tend to find solutions within a single homotopy class. Similarly, potential field planners consider only the basin of attraction containing the robot, thus selecting one homotopy class among many. If the chosen class later degrades in quality (perhaps due to sensor noise or dynamic obstacles), these algorithms have difficulty weighing the tradeoffs between further local optimization and switching to a new homotopy class.

An alternative hybrid approach by Howard [56] first generates a graph with regular discretization in the C-Space, then optimizes both nodes and edges in the graph to improve the quality of potential paths. Only after optimization completes is a discrete graph search performed. This approach elegantly balances the dual continuous/discrete choice, but this elegance comes at the cost of expensive optimization on many paths that will never be executed by the robot, thus potentially incurring significant up-front computation time prior to the onset of execution.

In this chapter, we introduce a planning algorithm that can simultaneously consider both decision-type and optimization-type choices and even reason about tradeoffs between the two types. The planner considers a set of locally-distinct routes—defined by an equivalence relation—that capture the notion of decision points. It selects among those routes and also selects paths within a route to optimize for safety, goal-directedness, and future flexibility in decision-making.

8.1 The Non-Monotonicity Problem

In Chapter 7, we show that local path equivalence can produce significantly more collision-free paths per unit time than traditional collision testing. In order for increased paths to translate into improved planner performance, a planner must exhibit *monotonicity*.

Definition 18 (monotonicity) Given a coarsely-sampled path set $\mathcal{C}$ and a densely-sampled path set $\mathcal{D}$ such that $\mathcal{C} \subset \mathcal{D}$, a monotone planner performs at least as well using $\mathcal{D}$ as it does using $\mathcal{C}$. Thus, we may say that under a monotonicity assumption, $\mathcal{D}$ dominates $\mathcal{C}$. □

Monotonicity (or its opposite) is a function not of the path set per se but of the planner and in particular of the heuristic function. In statistically-significant simulation experiments (see Section 5.3), we find that the basic minimum-distance heuristic function is non-monotonic. We believe this effect occurs due to the fact that $\mathcal{D}$ is expected to contain a more optimal path to the goal than $\mathcal{C}$—one that approaches closer to obstacles. In particular, $\mathcal{D}$ is likely to find risky narrow corridors that $\mathcal{C}$ might miss entirely. Therefore, we must establish that the additional paths contained in $\mathcal{D}$ can in fact be put to use in increasing navigation performance.

The architecture of the hierarchical planner is not amenable to the construction of a truly monotonic heuristic function. Suppose that a base path set is augmented with one extra path. In order to ascertain, during each replan iteration, whether the extra path should be selected in lieu of the otherwise preferred path, a monotonic planner must be able to anticipate what choices the heuristic function will make during successive cycles. The path chosen at each cycle has potential downstream effects without bound. Therefore, monotonicity requires the planner to determine that selecting the new path will not, for all time, result in a failure that would not otherwise have occurred. However, the design of the hierarchical planner abstracts away the high fidelity plan except in the immediate future. The global planner simulates robot motion at low fidelity and so does not possess the ability to predict the effects of selecting a particular path. Thus, any truly monotonic heuristic function would be impractical to implement in this context. We can, however, use path equivalence to produce a heuristic function more closely resembling the monotonic ideal.

Since the local planner has a limited horizon, the resulting planned route is a concatenation of paths from the local and global planners. Only the local paths are feasible to execute directly on the robot, so the local planner must replan at regular intervals to allow continued progress. Thus, Alg. 1, the Local Planner Algorithm, outputs a sequence of paths, the concatenation of the initial segments of which forms the true route. At the end of each replan cycle, the planner executes the beginning of the route representing the least cost to the goal, a heuristic known as *Best_Path*. Using this heuristic, traditional hierarchical planners produce strongly goal-directed behavior that comes with two drawbacks: temporal incoherence and excessive obstacle proximity.

8.2 The Temporal Incoherence Problem

Temporal incoherence occurs because the *Best_Path* heuristic does not generate consistent behavior between replan cycles, meaning that there is no deliberate process to maintain certain decisions throughout navigation. Often in hierarchical planning, the ultimate route executed by the local planner algorithm is an emergent behavior because the planner lacks any continuity of

intent between consecutive replan cycles. We propose local path equivalence as a means of representing such continuity. In choosing a sequence of local paths, local planners implicitly also select a sequence of equivalence classes. This observation provides another perspective in which to view local path equivalence: based on the limited information available to the local planner within a given replan cycle, the planned routes of all equivalent paths are homotopic. We propose a new algorithm to improve navigation performance by explicitly considering continuity within each replan cycle.

In general, we would like each replan cycle to select a new path that closely resembles the previous path, but such is not always the case. In Section 5.3, we proposed increasing the chance of such an outcome with the *Best_Path* heuristic by preserving the unexecuted remainder of the previous path as a **continuation** (Def. 6), which is considered along with the ordinary path set within subsequent cycles. Even so, on some occasions, consecutive replan cycles may switch equivalence classes, thus selecting a new planned route. *Best_Path* does not distinguish between classes, so such switches may happen arbitrarily often. Frequent switching is typically associated with perception noise. In especially noisy systems or where two planned routes are about equally costly, the planner may rapidly alternate between routes, thus effectively following an unplanned and undesirable path directly towards the obstacle separating the two routes.

8.3 The Obstacle Proximity Problem

Obstacle proximity, the second drawback incurred by *Best_Path*, risks the safety of the robot in cases of outside disturbance or internal prediction error. From a planning perspective, nearby obstacles also substantially reduce the quantity and diversity of safe paths available in subsequent replan cycles.

Two related approaches to the problem of decreasing robot proximity to obstacles have been in use for many years. The first approach involves "growing" the obstacles using a hard buffer, first proposed by Buhmann et al. [19], which runs the risk of closing off narrow openings. This problem is partially ameliorated by making the obstacle growth-radius vary in proportion to robot speed.

The second approach involves placing a soft buffer around each obstacle in the form of a gradient of elevated cost, such that cost varies inversely with obstacle proximity. This approach was first proposed by Thorpe [134]. Although this approach does not eliminate options from consideration, it is difficult to predict how a given cost function will affect decisions between corridors.

The drawback of both approaches is that they couple two distinct decisions: which route (equivalence class) to follow, and how to proceed (which path in the class) along that route. These decisions are of qualitatively different character because continual fine-tuning is possible throughout the traverse of a corridor, but the choice of corridor to be traversed requires a discrete decision that soon becomes irreversible (Fig. 8.1). To understand why this coupling is important, consider a decision between narrow and wide corridors. Clearly, the wide corridor is safer, but it may also be significantly more circuitous. There is a tradeoff between corridor width and path length, but both of the prior approaches handle the tradeoff in ways that are difficult to engineer or tune, thus leading to undesirable emergent behavior.

8.4 Improved Hierarchical Planner

We introduce a new multi-stage path selection algorithm that separates these two decisions, thus allowing them to be weighed separately and traded off against one another. This process in turn improves planning and control flexibility, increasing continuity of plans, and retaining goal-directedness. Through application of a set of rules based on path equivalence (applied both within and across replan time steps), the algorithm selects paths for execution that guide the robot sufficiently far from obstacles while moving consistently towards the goal.

8.5 Logical Succession Path Relation

In Chapter 7, we demonstrated the value of path equivalence in a single replan cycle. We now introduce a relation on path equivalence classes to detect logical succession across multiple replan cycles.

Definition 19 (logical succession) **Logical succession** is a strict partial ordering among equivalence classes $\mathcal{A} \rhd \mathcal{B}$ such that some paths $p_A \in \mathcal{A}$ and $p_B \in \mathcal{B}$ exist for which $\mu_H(p_A, p_B) \leq 1$ and $\mathcal{A}$ was generated in an earlier replan time step than $\mathcal{B}$. □

This definition establishes that two paths covering largely the same terrain but produced by different replan cycles can be said to follow the same route. The definition assumes a small time increment between replan cycles, such that little ground is covered in the interim. For larger steps, the definition would instead need to compare the end of p_A to the start of p_B. Of course, the pairwise logical succession property can be precomputed for our fixed path set in order to optimize performance.

In considering a new path for execution, logical succession provides a powerful tool for a planner to distinguish between paths that represent major and minor alterations to the prior plan. Suppose the planner just executed path p_i at time step $t-1$. We initially choose to consider at time t only those paths p_j such that $[p_i] \rhd [p_j]$, where $[p]$ describes the equivalence class containing p. This restriction provides continuity of plan. Often, each equivalence class has only one logical successor at the following replan time step. However, merges and splits may occur at critical points along the robot's traverse (Fig. 8.2). When the planner detects a split, it is important to select the branch that maximizes success, given the locally-available information.

8.6 Multistage Path Selection Algorithm

We introduce the multistage local planner algorithm (Alg. 8) to make principled path selections that trade off among the issues of logical succession, safety, and estimated path length to the goal. At a high level, the algorithm consists of two stages. Stage one selects for consideration a subset of $\mathcal{P}_{free}$ comprising one or more equivalence classes in order to ensure progress, safety, and consistency. Stage two selects from among the chosen subset one path for execution that trades off safety and cost of the path, while retaining goal-directedness.

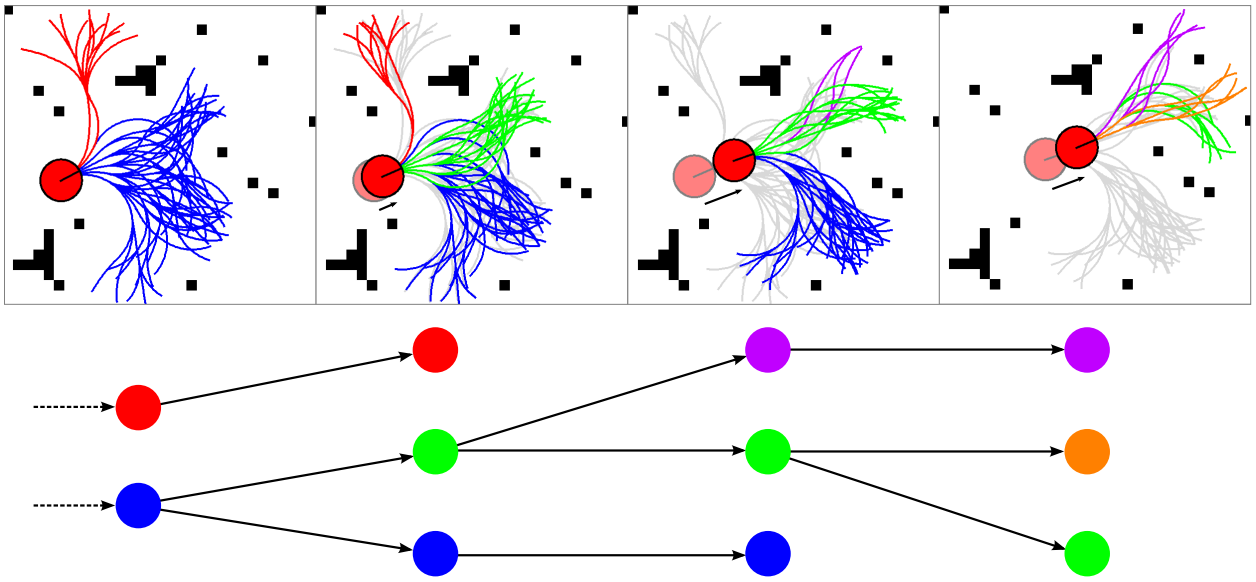


Figure 8.2 Between replan cycles, safe paths are associated by a logical succession of equivalence classes. This strict partial ordering relation is represented by a directed acyclic graph. Graph edges represent the relationship $\mathcal{P}_1 \rhd \mathcal{P}_2$. Graph node colors (and matching equivalence class path colors) are conserved in consecutive replan cycles only for the largest logical successor class. Between cycles, we may detect a termination, split, merge, or continuation of the previous equivalence classes. By preferring the logical successor of the previously-commanded path, subsequent path selections give better performance.

8.6.1 Stage One: Solving the Decision Problem

In generally preferring to execute a new path that is a logical successor to the previously chosen equivalence class, we largely eliminate sensitivity to noisy perception data. Two exceptions arise in which the algorithm will not execute a logical successor path. First, if a non-successor equivalence class predicts a significantly lower cost to the goal, then the planner switches classes on the assumption that the magnitude of the change exceeds that of likely perception noise. Second, we allow the algorithm to consider broader alternatives—whether more or less costly—if all logical successor classes terminate or become *narrow*.

Definition 20 (narrow) A **narrow** equivalence class contains few paths. We employ path count as a proxy for the measure of a corridor in path space. Thus, a low path count indicates little space to maneuver the robot through a narrow corridor. Non-narrow classes are called **wide**. □

We define a constant fraction, MIN_PATH_THRESH, which adaptively selects the cutoff in corridor width as a percent of the number of paths in $\mathcal{P}_{free}$. Thus, the more densely we sample the space of paths, proportionately more paths are required to constitute a wide corridor. Even when densely sampling the path space, a highly cluttered environment may eliminate all but a few paths through collision with obstacles. In such a case, a passage containing relatively few paths may still be to be considered "wide" in comparison to others with fewer paths.

This concept of wide and narrow corridors closely resembles that of Borenstein and Koren [15]. Their vector field histogram represents obstacle density projected down to one dimension corresponding to heading. Sparse regions of the histogram indicate corridors, but due to the

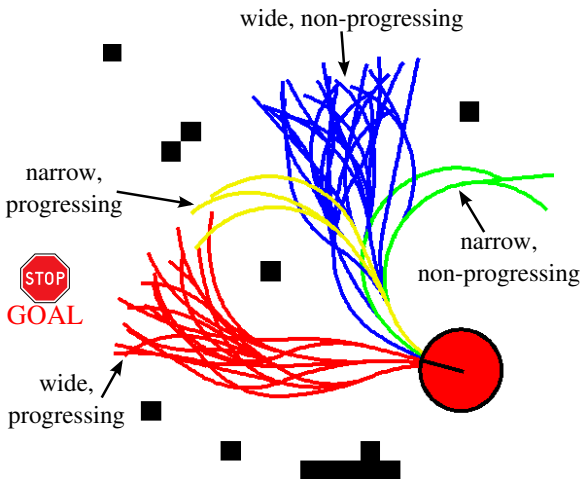


Figure 8.3 These four equivalence classes are annotated as to whether each is narrow or wide and progressing or non-progressing. Width is measured by the number of paths in the class as a fraction of all surviving paths, whereas progressivity describes whether its paths make progress towards the goal at left.

projection, only the component of corridor width perpendicular to that projection is recorded. Our approach to corridor detection and width estimation is more general since it closely approximates the full capabilities of the robot and is not limited to a particular obstacle configuration or observational perspective.

Although Alg. 8 displays a preference for wide logical successor classes, it strictly selects only progressing paths within a class for consideration.

Definition 21 (progressing) A **progressing** path is one for which both of the following two points are nearer to the goal than is the current robot position, according to the global planner:

1. the point one replan time step in the future, and
2. the end point of the local path.

Fig. 8.3 illustrates equivalences classes that are wide, narrow, progressing, and non-progressing. The progressing property is often shared by all paths in an equivalence class, but certain large classes in the absence of clutter can have mixed progressivity. By executing only progressing paths, we ensure that the robot monotonically approaches the goal, thus guaranteeing termination. Furthermore, by eliminating non-progressing paths during stage one, we are free to ignore goal-directedness in stage two while still guaranteeing progress.

When testing progressivity in a real implementation, it may be preferable to consider only criterion 2, a path's endpoint, and ignore the next step. Recognizing that curvature-constrained local paths need more space to maneuver than global grid paths, this relaxation provides the planner additional safety and flexibility when navigating around sharp corners, at the expense of termination guarantees.

In our implementation, Alg. 6 is used to eliminate nonprogressing paths and divide the rest according to the narrow/wide dichotomy. Alg. 8 uses it as a helper function in establishing an order of preference in selecting $\mathcal{S}$, the set of paths for consideration. The net order of preference is:

1. All wide, progressing, logical successor classes

Algorithm 6 $(\mathcal{W}, \mathcal{N}) \leftarrow$Divide_Wide_Narrow($C$, t)

Input: C – candidate set of classes; t – threshold size of class
Output: $\mathcal{W}$ – set of paths in wide classes; $\mathcal{N}$ – set of paths in narrow classes

```
W ← ∅
N ← ∅
for all C ∈ C do
    if |C| > t then
        W ← W ∪ C                                   // Paths in wide classes
    else
        N ← N ∪ C                                   // Paths in narrow classes
    end if
end for
W ← Cull_Nonprogressing_Paths(W)                    // Omit paths that move robot away from the goal
N ← Cull_Nonprogressing_Paths(N)
return (W, N)
```

Algorithm 7 $p \leftarrow$Optimize_Path(x, h, e, p)

Input: x – initial state;
h – a heuristic function for selecting a path to execute;
e – equivalence object
p – seed path for optimization
Output: Return a path similar to p but safer

```
repeat
    N ← e.Get_Neighbors(p) ∪ {p}
    p ← h.Farthest_Obstacle_Path(x, N)              // Select path in set farthest from nearest obstacle
until p converges or p.obstacle_proximity > 3/2 robot_diameter
return p
```

2. Any wide, progressing class
3. All narrow, progressing, logical successor classes
4. Any narrow, progressing class
5. Return failure

After choosing a preliminary $\mathcal{S}$, we must check if the planner has found a highly suboptimal subset of paths; the algorithm compares the best path in $\mathcal{S}$ against the best path in $\mathcal{P}_{free}$. A difference above a certain SCORE_THRESH provokes a mid-course correction. Such a switch of equivalence class should be a rare event.

In deciding when to override logical succession, we are making a choice with global implications based on unreliable information from a low-fidelity global planner. Lacking detailed knowledge of the complete path to the goal, we instead consider a calculation based solely on the statistics of this environment's average obstacle density, which predict that a narrow corridor

Algorithm 8 $(p, \mathcal{L})$ ←Multistage_Local_Planner_Algorithm(w, x, h, e, $\mathcal{P}$, $\mathcal{L}$)

Input: w – a costmap object;
x – initial state;
h – a heuristic function for selecting a path to execute;
e – equivalence object;
$\mathcal{P}$ – a fixed set of paths;
$\mathcal{L}$ – equivalence class of path selected in prior call (initially $\emptyset$)

Output: p – a path progressing safely toward the goal;
$\mathcal{L}$ – equivalence class of p

$\mathcal{P}_{free} \leftarrow Get\_Safe\_Progressing\_Paths(w, x, \mathcal{P})$ // May invoke implicit path test
$b \leftarrow h.Best\_Path(x, \mathcal{P}_{free})$ // Greedy shortest path
// Stage 1: select equivalence classes for consideration; trade off succession and corridor width
if $\mathcal{L} \neq \emptyset$ **then** // Compute successor path candidates
  $C \leftarrow e.Get\_Logical\_Successor\_Classes(\mathcal{P}_{free}, \mathcal{L})$ // Returns a set of classes
  $(\mathcal{W}_s, \mathcal{N}_s) \leftarrow Select\_Classes(C, \text{MIN\_PATH\_THRESH} \times |\mathcal{P}_{free}|)$
else
  $(\mathcal{W}_s, \mathcal{N}_s) \leftarrow (\emptyset, \emptyset)$
end if
$E \leftarrow e.Compute\_Equivalence\_Classes(\mathcal{P}_{free})$
$(\mathcal{W}, \mathcal{N}) \leftarrow Select\_Classes(E, \text{MIN\_PATH\_THRESH} \times |\mathcal{P}_{free}|)$ // Non-successor classes
if $\mathcal{W}_s \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{W}_s$ // Consider the set of wide successor classes if some exist
else if $\mathcal{W} \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{W}$ // Prefer any wide, progressing class over a narrow successor
else if $\mathcal{N}_s \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{N}_s$ // Narrow successors are better than other narrow classes
else if $\mathcal{N} \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{N}$ // Last resort: take any path
else
  return failure
end if
$p \leftarrow h.Best\_Path(x, \mathcal{S})$
if $p.score - b.score >$ SCORE_THRESH **then**
  $\mathcal{S} \leftarrow \mathcal{P}_{free}$ // Jump equivalence classes to a significantly shorter route
end if //
// Stage 2: Select one path from the set, trading off path length with safety
$p \leftarrow h.Best\_Path(x, \mathcal{S})$
$p \leftarrow Optimize\_Path(x, h, e, p)$ // Find safe enough path in selected path set
$\mathcal{L} \leftarrow e.Class\_Of(p)$
return $(p, \mathcal{L})$

"pinch point" should occur periodically at some frequency during traversal. Given a distance remaining to reach the goal, we can estimate an expected number of risky narrow corridors remaining. SCORE_THRESH should be chosen so that the decreased risk (stemming from the shorter path length to the goal) of getting stuck in a future narrow corridor outweighs the immediate risk involved in the current route change, which may itself jump to a narrower corridor.

8.6.2 Stage Two: Solving the Optimization Problem

After establishing a final set of candidate paths $\mathcal{S}$, the algorithm moves on to stage two, which selects a single path for execution. Initially, it finds the greedy *Best_Path* option in $\mathcal{S}$, but this path may come unsafely close to an obstacle. Within the equivalence class containing the shortest path, the subroutine *Optimize_Path* performs a local, gradient-descent-type optimization in path space by traversing the equivalence graph. This optimization, which generates a soft safety buffer around obstacles, seeks to maximize the distance to the one nearest obstacle as described in Alg. 7.

In especially wide corridors, the robot should be free to follow a reasonably short path, so the obstacle proximity penalty decays to zero beyond 1.5 robot diameters. The proximity penalty function is only defined with respect to the one nearest obstacle, so in a narrow corridor the penalty is locally minimized by the path most nearly following the center of the corridor, thus maximizing both safety and future planning options. Note that the algorithm will follow even an extremely narrow corridor, provided that the route represents the best means to progress towards the goal.

Ultimately, the algorithm we describe here improves on the original local planner algorithm by executing an action that is safe, maximizes future planning/control options, and remains consistent across replan cycles, all while retaining goal-directedness.

8.7 Experimental Results

We tested the multistage local planner algorithm (Alg. 8) over a variety of environments at a range of obstacle densities in order to evaluate the effects on planner performance of awareness of local equivalence classes. Experimental setup was the same as in Section 7.3, except that here we tested the planner on 500 different planning problems for each obstacle density. Replan cycle time for these experiments is 0.1 sec.

Fig. 8.4 shows success rate for the local planner algorithm (greedy) and multistage local planner algorithm (equivalence aware). The latter produces a statistically significant improvement of 7.6% at solving planning problems in dense clutter. Path length increases only negligibly, and despite the extra path length, we find a decreased path cost, expressed as

$$c(p) = \int_p \frac{ds}{od(p(s))}, \tag{8.1}$$

where $od(s)$ is the distance to the nearest obstacle from the given point along the path. Fig. 8.5 shows the mean absolute cost for each planner averaged over all runs, while Fig. 8.6 shows the relative cost between the two planners. The overall mean change, shown with the solid red line,

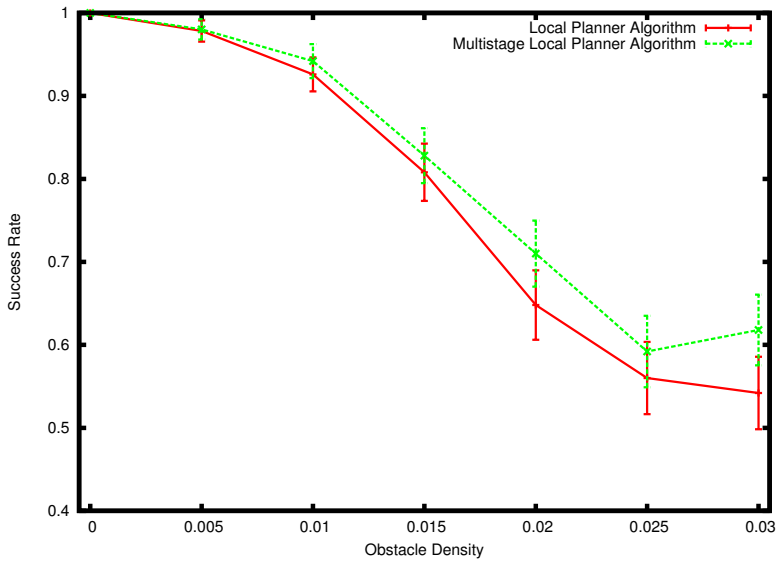


Figure 8.4 Improvement in overall planner success rate. At high clutter, Alg. 8 significantly outperforms Alg. 1. Note that 0.03 obstacle density in the workspace corresponds to approximately 67% C-space density.

indicates that the robot stays about twice as far from obstacles. We compared performance on each test problem separately so the individual data-points (shown with blue crosses) represent cost improvement normalized to the difficulty of the problem.

In contrast to the overall trend, a small fraction of the experiments showed cost worsening. In roughly 2% of cases, skewed heavily towards the less dense environments, obstacle proximity increased with the equivalence-aware planner. We attribute this phenomenon to the interplay between choice and lookahead distance. In more open environments, the increased choice makes it more tempting to stray from continuity of plan. At a different lookahead distance, the planner may have understood its choices better, but there are of course significant costs to increased lookahead. Making this tradeoff dynamically presents an interesting opportunity for future work.

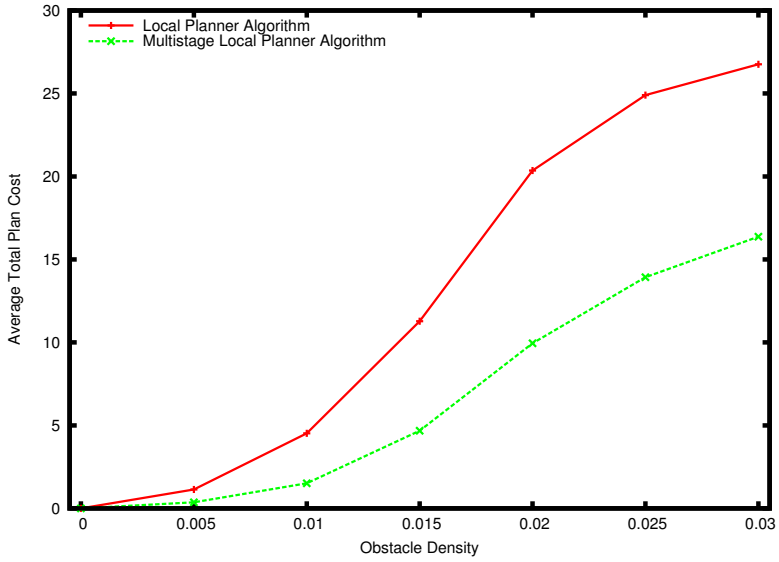


Figure 8.5 Absolute cost function penalizing obstacle proximity and path length.

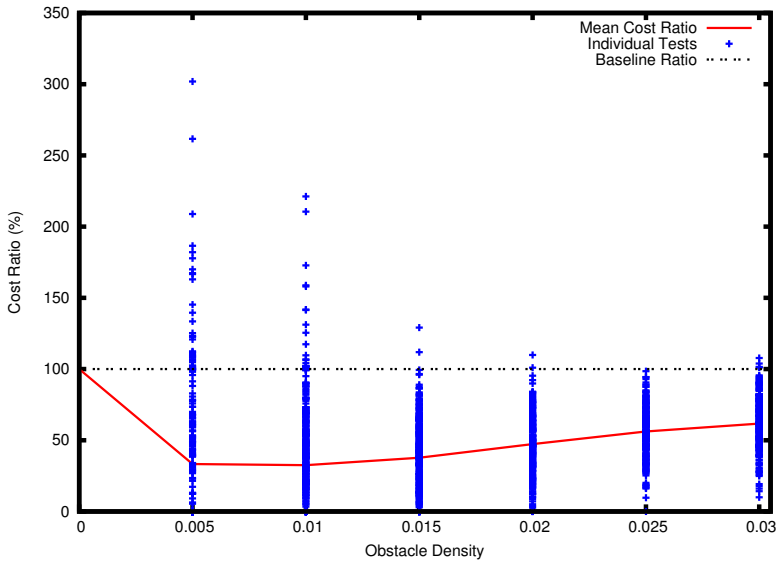


Figure 8.6 Ratio of *Best_Path* cost to multistage local planner algorithm cost. Lower is better. The cost of an individual path from either planner represents the path integral of the reciprocal of obstacle proximity. The overall mean cost improvement (solid red line) is about a factor of two, meaning that during navigation the robot stays twice as far from obstacles, on average. The scatter plot (blue crosses) shows the individual results for the 1838 experiments in which both planners successfully reached the goal.

Chapter 9

Extensions and Future Work

Thus far, we have examined the role that information reuse can play in the basic motion planning process. Many of our examples employ simple models such as a mobile robot in the plane. We now briefly turn to more complex systems and applications. We also discuss a number of other avenues for future work in model-based hierarchical planning.

9.1 Path Sampling

In this thesis, we introduce a real-time adaptive path sampler that is informed by the outcome of previous path tests. In the future, we plan to extend the planner to remember the locality model between consecutive replan cycles in order to spare it from relearning the model each time. We also note that we are disregarding the parts of a colliding path that test collision-free, which would provide additional data to the locality model.

As we mentioned earlier, there is a tradeoff between depth and breadth of search in path set design. The relationship between these traits and planning performance deserves significant study. Adaptation of the path set, either online or offline, to adjust this tradeoff for performance, is a promising area for future work.

More ambitiously, we remind the reader of our observation that the costmap is not an effective data format. A square grid is inefficient for the planner, which must integrate paths across many grid cells to compute the safety and desirability of paths. Neither is the costmap an efficient representation for the perception system, since sensors do not typically produce raw data in a Cartesian format. We believe that a more suitable format of costmap could potentially revolutionize robotics. After all, the planner spends the majority of its time essentially decoding the costmap. Even more promising, we hypothesize that the costmap could be circumvented entirely by putting perception data into a path space representation, thereby enabling the planner to directly discover which paths are safe.

9.2 Mobile Manipulation

This thesis describes preliminary work on hierarchical planning for mobile manipulation. We have reported results on a real robot for the decoupled navigation and arm planning problems.

We also demonstrated the combined mobile manipulation planner in simulation (Fig. 3.5). A thorough examination of the coupled mobile manipulation planner running on HERB remains a subject of future work. However, several other areas of future work present opportunities.

Bidirectional planners have been popular for a decade, particularly in the RRT community. In interleaved planning and execution, bidirectionality has a different meaning since the robot can only execute in the forward direction. However, we are already getting some benefit from bidirectional planning, as we are sampling pullback goals (Section 3.3.1), from which we know how to reach the ultimate goal. We believe this idea can be generalized to discover regions of space from which a goal state is most reachable.

At present, the hand approaches a goal position in any arbitrary orientation, but we could sample in fewer dimensions and constrain the palm or wrist to point towards a goal or other nearby object. Then, as the hand moves closer to its goal, we can use distally-mounted cameras to localize the target, providing improved position estimates while planning.

Finally, we would like to explore the planner's failure modes in hopes of designing a global planner which succeeds more often. Occasional failure is a fair price to pay for a planner that works quickly and generates reasonable paths under most circumstances. In addition to these benefits, the hierarchical planner offers reactivity to dynamic obstacles and nimble maneuvering in cluttered environments.

9.3 A Rigid Body in 3D

Next, we address the application of local path equivalence to a rigid body in a 3D workspace, such as an aircraft or spacecraft. Let us suppose this rigid body takes the form of a sphere (or can be approximated as one). Two different paths, no matter how close together, are never sufficient to establish path equivalence, as they were in two dimensions. Fig. 9.1 illustrates this fact. In three dimensions, three paths are equivalent if each of their pairwise Hausdorff separations is less than the *radius* of the robot. Fig. 9.2 depicts such a configuration of three paths and also shows how to construct a continuous deformation between any pair. As in the 2D case, we can perform implicit collision testing on a variety of paths. The region in which a fourth path must reside to implicitly declare it collision-free is shown in Fig. 9.2.

One who is familiar with topology might question the value of path equivalence in three dimensions because ordinary bounded obstacles do not induce additional homotopy classes. However, this is where local path equivalence really shines. Since the planner has only local knowledge, it cannot distinguish between a finite, long, skinny obstacle and an infinite, skinny obstacle. For all practical purposes, the finite obstacle might as well be infinite.

9.4 Variable-Size Robots in 2D

In comparison to (7.1), we similarly define a normalized Hausdorff metric as

$$\mu_H(p_i, p_j, d) = \inf_{\varepsilon}\{p_i \subset (p_j)_{\varepsilon d} \text{ and } p_j \subset (p_i)_{\varepsilon d}\}. \tag{9.1}$$

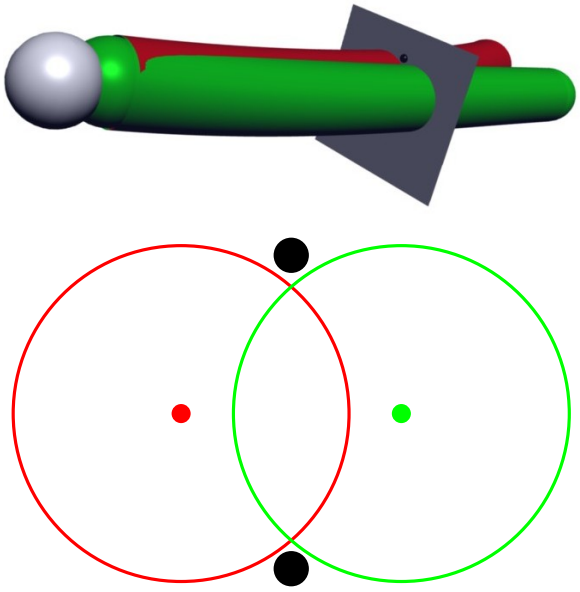


Figure 9.1 Two paths (red and green) of a spherical robot (gray) in a 3D workspace are insufficient to establish equivalence. The bottom figure depicts a cross-sectional slice through the swept volumes of the paths. Although two paths may be separated by less than one robot diameter, obstacles (black) may still prevent a continuous deformation between the paths.

We now introduce a variant of the basic 2D mobile robot in which the robot's size—still approximated as a disc—varies as a function of path length. This scenario occurs in the mobile manipulation problem, in which a mounted manipulator arm may extend out beyond the robot's own footprint. For example, the elastic strips of Brock and Khatib [18], when projected onto the floor, closely resemble this variable-width robot. A radius function $r_i(\ell)$ expresses the robot's radius with respect to point $p_i(\ell)$ along path p_i. We introduce the concept of proportional dilation, in which the path width grows in accordance with its radius function:

$$(p)_{r_i} = \{t \in \mathbb{R}^2 \colon \|p(\ell) - t\|_{L2} \leq r_i(\ell) \text{ for some } \ell \in I_L\}, \tag{9.2}$$

where $I_L = [0, L]$, with L the path length. Now a pair of nearby points on two neighboring paths may possess distinct diameters, as in Fig. 9.3, thus giving rise to a new normalized Hausdorff metric,

$$\begin{aligned}\mu_H^v(p_i, p_j, r_i, r_j) = \max(&\operatorname*{argmin}_{\varepsilon} (\forall \ell_a \in I_L, \exists \ell_b \in I_L \colon (p_i(\ell_a))_{\varepsilon r_i(\ell_a)} \cap (p_j(\ell_b))_{\varepsilon r_j(\ell_b)} \neq \emptyset),\\ &\operatorname*{argmin}_{\varepsilon} (\forall \ell_a \in I_L, \exists \ell_b \in I_L \colon (p_j(\ell_a))_{\varepsilon r_j(\ell_a)} \cap (p_i(\ell_b))_{\varepsilon r_i(\ell_b)} \neq \emptyset)).\end{aligned} \tag{9.3}$$

Intuitively, this variant of the Hausdorff metric finds the minimal scale factor for the two paths' radius functions such that no gap remains between the two paths following proportional dilation.

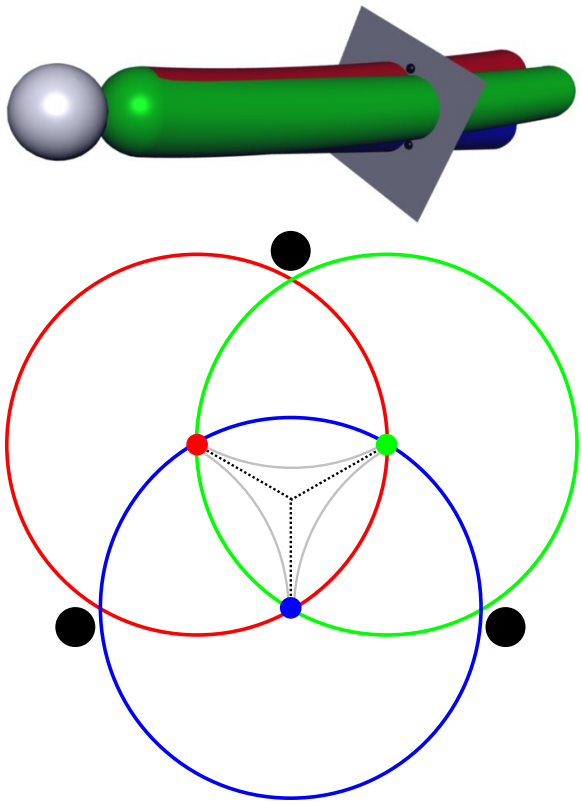


Figure 9.2 Three paths (red, green, and blue) are required to establish path equivalence in 3D. The swept volumes of the three paths are shown. To establish equivalence, we require that the Hausdorff distance between each pair of paths must not exceed the robot's radius. Given such proximity, one path may be continuously deformed to another by following the dotted lines, without risk of intersecting the black obstacles. Implicit collision testing may be performed on any fourth path discovered to be entirely inside the gray star-shaped region at center.

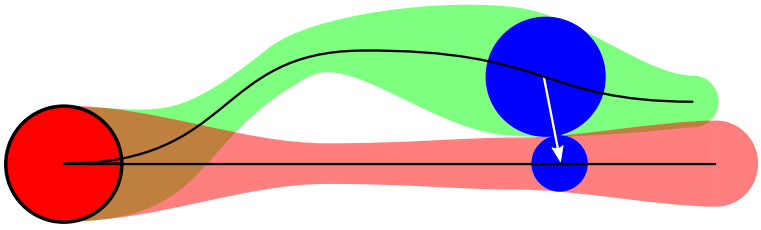


Figure 9.3 Mobile robot paths with variable radius. The nearest point on the opposite path depends on both the position and radius of each point along the path.

Following dilation, each point on either path is replaced by a disc. In order to ensure the above condition, each disc on one path must intersect some disc on the opposite path.

Given some appropriate constraints on the shape of paths as well as their radius functions, the equivalence relation on paths of variably-sized mobile robots then follows directly from the fixed-diameter case:

$$\mu_H^v(p_1, p_2, r_1, r_2) \leq 1 \implies p_1 \sim p_2. \tag{9.4}$$

Here, curvature bounds must apply to the boundary of the variable-diameter robot swath (Def. 3) in addition to the path itself. However, in the case of a mobile manipulator, where the reach of an arm is on the same order of magnitude as the mobile base diameter, this distinction is rarely critical.

9.5 Articulated Robots

We similarly apply path classification to the trajectories of manipulator arms in 3D. At a high level, the situation closely parallels the 2D mobile robot case we present in Section 7.1. Two paths, separated by at most the diameter of the arm, are equivalent under certain shape and proximity constraints. In contrast to a 2D rigid body, the central axis of the arm sweeps out a 2D manifold in the workspace, so points along our path are now parametrized by (s, ℓ), where $s \in I_S = [0, s_f]$, a distance along the arm's motion in the configuration space, and $\ell \in I_L = [0, L]$, a distance along the axis of the arm from base to end-effector. Thus, $p(s, \ell)$ corresponds to a particular workspace location along the arm's axis while the arm is in a certain configuration. A function $r(\ell)$ describes the radius of a disc circumscribing a cross-section of the arm at a point along its length. Note that radius is now a function function of arm length rather than trajectory position. Note also that the disc is now normal to the arm axis, whereas in the 2D shape-changing robot, it is coplanar with the path. Though nearly identical to (9.2), we define proportional dilation in the context of a two-parameter path function,

$$(p)_{r_i} = \{t \in \mathbb{R}^2 \colon \|p(s, \ell) - t\|_{L2} \leq r_i(\ell) \text{ for some } s \in I_s \text{ and } \ell \in I_L\}. \tag{9.5}$$

As in Section 9.4, we define a variant of the Hausdorff distance in the context of arm paths,

$$\begin{aligned}\mu_H^a(p_i, p_j, r_i, r_j) = \max(&\operatorname*{argmin}_{\varepsilon} (\forall (s_a l \ell_a), \exists (s_b, \ell_b) \colon (p_i(s_a, \ell_a))_{\varepsilon r_i(\ell_a)} \cap (p_j(s_b, \ell_b))_{\varepsilon r_j(\ell_b)} \neq \emptyset),\\ &\operatorname*{argmin}_{\varepsilon} (\forall (s_a, \ell_a), \exists (s_b, \ell_b) \colon (p_j(s_a, \ell_a))_{\varepsilon r_j(\ell_a)} \cap (p_i(s_b, \ell_b))_{\varepsilon r_i(\ell_b)} \neq \emptyset)),\end{aligned}$$

where $s_a, s_b \in I_S$ and $\ell_a, \ell_b \in I_L$. Note that this is a conservative expression for the distance separating two arm trajectories.

In considering the possibility that an obstacle divides two arm trajectories, the semantics of the application come into play. For example, objects in human spaces do not levitate, so in the absence of highly dynamic objects such as a thrown ball, we may relax the tight constraints imposed by by μ_H^a. Instead, the two arm trajectories need only to completely surround a pocket of space, meaning that their end-effector trajectories and end states overlap.

Next, we address constraints on arm path shape and length. Such concepts are inherently much more nebulous than their mobile-robot equivalents due to the arm's articulation—especially

for arms with revolute joints. It is difficult to pin down general, meaningful constraints on arm path length and shape.

In principle, a useful measure of path length could be obtained by computing swept volume of the arm. After all, in the case of a rigid body mobile robot, all swaths of a given length that do not cross over themselves have equal length. In the case of manipulator arms (especially those with revolute joints), many useful motions do involve swept volumes that "cross over themselves," so an alternate formulation of path length is needed. Just as the length of a mobile robot path is found by integrating velocity, the length of an arm path may be found by integrating a form of velocity as well. Given an arm path p_i executed with unit C-Space speed, let $v_i(t, \ell)$ be the workspace velocity of point $p_i(t, \ell)$ along the axis of the arm at time t. We propose two alternative path length measures:

$$L_{mean}(i) = \frac{1}{L} \int_0^L \int_0^{s_f} v_i(s, \ell)\, ds\, d\ell \tag{9.6}$$

$$L_{max}(i) = \int_0^{s_f} \max_{\ell \in I_L} v_i(s, \ell)\, ds \tag{9.7}$$

In the case of a rigid body under arbitrary motion in $\mathbb{R}^3$, the mean and max path length are always related by a factor between one and two. For an articulated chain, the factor may be greater.

We now move on to address path shape constraints for an arm. This issue is both complex and mechanism-dependent. In previous work [84], we utilized path sets comprising straight lines in C-Space. Of course, such "straight" trajectories can involve arbitrarily high curvature of some point on the arm within the workspace. Given a sampling of such paths dense enough to establish path equivalence, it is not clear that the planner would often discover multiple equivalence classes. It would therefore be left to a given manipulation application to further constrain path sampling to a set of tasks useful for a given problem.

A few approaches worth exploring further include bounding the energy consumed in executing a given path (after subtracting out torques associated with gravity compensation), and retraction-based approaches. In the latter case, we propose to compare paths by utilizing retraction-like reductions in dimensionality of a search space, such as those proposed by Choset and Burdick [29], Sun and Lumelsky [132]. Under this reduction, any given path in the free configuration space maps to a path in a one-dimensional set, which is the deformation retract of the freespace. We can then employ such retracts as a graph-like roadmap and compare only paths whose corresponding graph paths are similar. For general articulated systems, this approach raises some challenges of its own, such as the fact that in three or more dimensions, these one-dimensional retracts are necessarily either not connected or have extra loops not corresponding to topological features of the original freespace [30].

Despite these challenges, our path equivalence relation holds promise in the domain of motion planning for articulated robots, such as manipulator arms. For instance, even lacking constraints on path shape, it is possible to utilize μ_H^a to accelerate collision testing. Although arms pose greater challenges for satisfying the necessary conditions on proximity and betweenness, the cost of each collision test is significantly higher for articulated robots than it is for rigid bodies, so the gains remain potentially significant.

9.6 Steerable Needles

Steerable needles represent an application of local path equivalence to medical technology. Steerable needles are long, flexible, hollow needles with a bevel tip. During insertion, the bevel tip causes the needle to follow a constant curvature path. The shaft of the needle may be twisted to change the direction of curvature. Such needles can be used in medical procedures to reach soft tissue anatomical features that would otherwise be inaccessible due to obstructing anatomy of a hard (bone) or delicate (nerve, artery) nature. Steerable needles are interesting from a motion planning perspective because they are underactuated, having only two velocity controls. One may vary the rate of insertion and the rate of rotation about the axis of the needle. This twisting motion alters the plane in which the needle bends [143].

During needle motion, uncertainty is introduced to the path primarily in the form of a random variable added to the needle's curvature. This uncertainty derives from the complexities of interaction with human tissue. Instantaneously, the uncertainty in position increases in the direction of the vector normal to the needle within the plane of curvature. Meanwhile, the instantaneous control inputs act primarily in vectors along the needle and normal to the plane of curvature. By duty-cycling the needle, steering control can be applied to cancel out errors in curvature, thus the needle can track arbitrary paths of bounded curvature [101].

Planners have been proposed for needle steering to account for uncertainty in both motion [2] and localization [137]. It should be possible to extend many steerable needle planners using path equivalence. The equivalence algorithms in this thesis assume that any path under consideration may be chosen deterministically. Thus, the distribution of paths is purely a function of control inputs. In the steerable needles context, the distribution among paths is generated by a combination of control and uncertainty. Consequently, we must find equivalence between entire groups of paths across a probability distribution instead of between individual paths. Thus, steerable needles constitute both a theoretical extension and a promising application for future work.

Chapter 10

Conclusion

This thesis focuses on the problems of generating, testing, and selecting candidate control-space trajectories in real time in a local planner. High performance planning tasks demand predictive models in order to generate feasible, collision-free workspace trajectories. Such feasible trajectories can be followed without any kind of smoothing or post-processing. The challenge inherent in model predictive planning is that the model maps controls to configuration space trajectories, whereas path diversity is measured in the path space. Thus, the path diversity problem must account for many paths passing through many configurations over time. We overcome this challenge through a combination of offline precomputation, clever online information management, and a hierarchy of planners at varying levels of model fidelity, thus permitting the planner to defer some computation until it is needed.

In all stages of motion planning, we exploit latent, freely available information to produce measurable gains in planning performance. By several different methods, we are reusing information previously generated or discovered. However, the key to realizing performance gains from such information is representational. By storing information in the immediate form consumed by the planner—a path space representation—we realize a substantial computational savings. And by maintaining only the data most relevant to the current planning problem, this path space representation reduces the cost of search both in the form of collision testing and path exploration.

In this thesis, we leverage path space representations of several forms. First, we perform diverse path set generation offline and augment it with adaptive online path set generation in order to increase the yield of high-quality, diverse path samples. By leveraging previous collision tests and the principle of locality, we develop a model that predicts path test outcomes using path space relationships. This model runs fast enough to "pre-test" many paths and select promising candidate paths combining the properties of survival and diversity.

We also introduce a purely local path equivalence relation with applications in path testing and path selection. We provide a method to rapidly identify equivalence among path samples in order to classify those paths that survive a collision test. Given two members of the same class, we show that a continuous deformation between them exists, which is both collision-free and feasible. By recognizing that an untested path is continuously deformable to two other equivalent, tested paths, the untested path may be "implicitly" tested, declaring it safe without an expensive, explicit test. We show that up to 90% of paths may be implicitly tested, delivering

speedups of more than 300% in collision testing performance.

We also exploit the classification of safe paths during the path selection phase. Local path classes afford the planner an opportunity to defer some choices until later in executing a path. While the robot must commit to its first step prior to the onset of motion, it retains the flexibility to optimize the unexecuted remainder of the path during future replan iterations. By executing a path from a large equivalence class, the planner not only increases safety but also ensures greater choice in the future, thus enhancing the robot's ability to deal with unforeseen contingencies, all while retaining goal-directedness.

This thesis explores themes of accelerating computation in motion planning. Using approaches presented here, a planner may provide more safe, feasible, diverse paths per unit time, and it may select more intelligently among them. However, additional paths generated display diminishing returns in corresponding plan quality.

The argument is often made that, due to improvements in processor speed, motion planning performance accelerates for free with the passage of time. In the near term future, collision testing will remain a majority of total planning time, and these approaches multiply any hardware performance gains, thus retaining their advantage. In the more distant future, computing technology may reach the advanced state in which a planner can generate and test a sufficiently dense selection of paths by brute force alone while meeting realtime constraints. We show in this thesis that the benefits of the multistage path selection algorithm actually increase as more paths are tested. Thus, our contributions to heuristic path selection (specifically, in understanding the set of discrete decisions facing a robot amongst obstacles) will likely have the greatest long-term impact.

In addition, these algorithms generalize to a variety of scenarios and environments. We make no assumptions about the robot's dynamics or kinematics apart from determinism. We actually benefit from added constraints that generate a meaningful form of path equivalence. Although we depict robots with a ball shape, any shape is allowable, including articulated robots of arbitrary shape. Another area of generalization comes in the context of path set frame. This work is presented in the robot-fixed frame for ease of presentation. However, all of the approaches discussed in this thesis apply equally well to world-fixed planners. A planner such as RRT-Blossom [69] would benefit equally well from many of these approaches. Likewise, our algorithms apply to path sets comprising independent paths as well as path trees.

10.1 Contributions

- **Model based planners.** This thesis contributes to model based planners, which have been employed in diverse environments, from office settings to hospitals to on-road driving to unstructured rough terrain. By improving the ability of such planners to generate, test, and select sets of diverse candidate paths, robots' performance and responsiveness to their environment will be increased.
- **New mobile manipulation approach.** We present a new approach to decoupling planning in the high-dimensional configuration space of motion planning for mobile manipulation. This algorithm works effectively and with low latency in common environments.
- **Information reuse.** We propose a set of practical algorithms for improving path set performance at all stages of the planning process by exploiting readily available information that has already been computed by the planner.
- **Adaptive path sampling.** We present a real-time algorithm to feed back path test results in order to sample a diverse set of paths high likelihood of surviving the collision test. This algorithm takes advantage of the principle of locality to select a combination of paths from two classes: maximum-entropy paths optimize exploration of the locality model, while maximum-safety paths optimize exploitation.
- **Implicit collision test.** Using the local path equivalence property, we efficiently detect when the entire swath of a path under test has been previously tested by other surviving paths. In such cases, the path can be declared implicitly safe without performing an explicit path test. Since path testing occupies a significant majority of overall planning time, implicit path testing dramatically increases planning performance. We show an increase of 300% in safe paths generated per unit time.
- **Multistage path selection.** In order to take advantage of increased safe path output, we present an improved path selection algorithm in two stages. In stage one, one or more equivalence classes of paths are selected based on logical succession, corridor width, and progressivity. Having selected only paths that make progress toward the goal, in stage two, an individual path is selected for execution without regard to goal-directedness. Doing so improves the safety of selected paths while ensuring forward progress. We show that the multistage path selection algorithm eliminates the non-monotonicity problem and tempers excessive goal-directedness.
- **Empirical evaluation.** We utilize several metrics to analyze the performance of path sets and planning algorithms. We perform large quantities of tests in simulation in order to produce statistically significant results.
- **The principle of locality.** We examine the principle of locality, which states that points near known obstacle collision points tend to also be in collision with the same obstacle. We propose a set of locality models to efficiently summarize and feed back obstacle information that is most valuable to the planner in sampling future paths.
- **Information management.** Given the vast quantity of information available to a robot through sensors and prior data, the management of that information is often a challenge.

We efficiently manage information, retaining only that which is most valuable to the future planning process. Furthermore, we maintain data structures to keep the information in an immediately usable form for the planner.

- **Local path equivalence.** We introduce a new form of equivalence relation called local path equivalence. This relation, based on a continuous deformation between paths, maps a large set of scattered paths into a small set of contiguous regions free of obstacles. From the context of a planner, these regions represent navigation alternatives. Thus, local path equivalence provides the planner with a sophisticated tool for understanding the true route options in highly-cluttered environments.
- **Diverse costmaps.** These algorithms address a variety of planner requirements, including continuous-valued cost maps and binary cost maps.
- **Precomputation.** By utilizing fixed, precomputed, input-sampled path sets, we are able to guarantee diversity under any possible initial state by precomputation. Furthermore, we precompute many properties of these paths, such as metrics, betweenness, and nearest neighbor queries.
- **Efficient algorithms.** All of the algorithms presented here are of polynomial complexity with respect to path count and run in real time. Efficient run-time is the key to achieving speedups throughout the planning process.

Bibliography

[1] T. Allen, J. Underwood, and S. Scheding. A path planning system for autonomous ground vehicles operating in unstructured dynamic environments. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2007. 2.2

[2] R. Alterovitz, M. Branicky, and K. Goldberg. Motion planning under uncertainty for image-guided medical needle steering. *International Journal of Robotics Research*, 27 (11–12):1361–1374, 2008. 9.6

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, Houston, TX, USA, March 1998. 6.2

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: combinatorial optimization problems and their approximability properties*, pages 419–420. Springer, 1999. 4.2, 5.1

[5] A. Autere. Hierarchical a* based path planning–a case study. *Knowledge-Based Systems*, 15(1):53–66, 2002. 2.2

[6] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-3-4):121–155, 1993. 7, 8

[7] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2–4):121–155, 1993. 4.3, 5.5

[8] R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60:503–516, 1954. 2.5

[9] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *Proceedings of the International Conference on Robotics and Automation*, May 2009. 3.3.4

[10] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet Romea, and J. Kuffner. Manipulation planning with workspace goal regions. In *Proceedings of the International Conference on Robotics and Automation*, May 2009. 3.3.1

[11] W. H. Beyer, editor. *CRC Standard Mathematical Tables and Formulae*. CRC Press, Boca Raton, FL, 1991. 6.5.1

[12] S. Bhattacharya, V. Kumar, and M. Likhachev. Search-based path planning with homotopy class constraints. In *Proceedings of the National Conference on Artificial Intelligence*, 2010. 7

[13] H. Blum. A transformation for extracting new descriptors of shape. In Weiant Whaters-Dunn, editor, *Proceedings of the Symposium on Models for the Perception of Speech and Visual Form*, pages 362–380, Cambridge, Mass., 1967. MIT Press. 1

[14] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield. Little ben: The ben franklin racing team's entry in the 2007 DARPA urban challenge. *Journal of Field Robotics*, 25(9):598–614, 2008. 5

[15] J. Borenstein and Y Koren. The vector field histogram—fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991. 7, 8.6.1

[16] M.S. Branicky, R.A. Knepper, and J.J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *Proceedings of the International Conference on Robotics and Automation*, Pasadena, CA, 2008. 5.2

[17] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052, December 2002. 2.3, 5.5.3

[18] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052, December 2002. 8, 9.4

[19] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. E. Schneider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, 1995. 5, 8.3

[20] B. Burns and O. Brock. Information theoretic construction of probabilistic roadmaps. In *Proceedings of the International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, October 2003. 6.2

[21] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proceedings of the International Conference on Robotics and Automation*, Barcelona, Spain, April 2005. 6.2

[22] B. Burns and O. Brock. Single-query entropy-guided path planning. In *Proceedings of the International Conference on Robotics and Automation*, Barcelona, Spain, April 2005. 6.2

[23] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Proceedings of Robotics: Science and Systems*, Cambridge, MA, USA, June 2005. 6.2, 6.4, 6.6

[24] E. F. Camacho and C. Bordons. *Model predictive control*. Springer Verlag, 2004. 2.4

[25] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28:104–126, 2009. 3.4

[26] S. Candido, Y. Kim, and S. Hutchinson. An improved hierarchical motion planner for humanoid robots. In *Proceedings of the Internaional Conference on Humanoid Robots*, Daejeon, Korea, December 2008. 2.2

[27] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *Proceedings International Conference on Intelligent Robots and Systems*, pages 43–48, 2001. 2.4

[28] J. Choi and E. Amir. Combining planning and motion planning. In *Proceedings of the International Conference on Robotics and Automation*, Kobe, Japan, May 2009. 3.4

[29] H. Choset and J. Burdick. Sensor based planning, part I: The generalized Voronoi graph. In *Proceedings of the International Conference on Robotics and Automation*, pages 1649–1655, 1995. 2.5, 7, 8, 9.5

[30] H. Choset and A. A. Rizzi. Topology in motion planning. In *Proceedings of the International Symposium on Robotics Research*, August 2005. 9.5

[31] M. Daily, J. Harris, D. Keirsey, D. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong. Autonomous cross-country navigation with the ALV. In *Proceedings of the International Conference on Robotics and Automation*, Philadelphia, PA, 1988. 2.2

[32] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, July 2008. 3.3

[33] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 2.5

[34] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous driving in unknown environments. In *Proceedings of the International Symposium on Experimental Robotics*, Athens, Greece, July 2008. 2.5, 3.3.3

[35] D.D. Dunlap, C.V. Caldwell, and E.G. Collins. Sampling-based model predictive control. In *Proceedings of the American Control Conference*, Baltimore, MD, June 2010. 2.4, 5.5

[36] L.H. Erickson and S.M. LaValle. Survivability: Measuring and ensuring path diversity. In *Proceedings of the International Conference on Robotics and Automation*, Kobe, Japan, May 2009. 4.2, 4.2, 5.2

[37] M. Farber. Topological complexity of motion planning. *Discrete & Computational Geometry*, 29(2):211–221, 2003. 7

[38] D. Ferguson and A. Stentz. Field D*: An interpolation-based path planner and replanner. In *Proceedings of the International Symposium on Robotics Research*, October 2005. 2.2, 2.5, 5.2

[39] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971. 2.2

[40] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997. 2.2

[41] E. Frazzoli, M. A. Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1), January–February 2002. 5.5.2

[42] E. Frazzoli, M. A. Dahleh, and Eric Feron. Maneuver-based motion planning for nonlinear

systems with symmetries. *IEEE Transactions on Robotics*, 21(6), December 2005. 5.2

[43] M. Garber and M. C. Lin. Constraint-based motion planning using voronoi diagrams. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, pages 541–558, 2002. 2.5

[44] N. H. Gardiol and L. P. Kaelbling. Action-space partitioning for planning. In *Proceedings of the National Conference on Artificial Intelligence*, Vancouver, Canada, 2007. 7

[45] R. Alami R. Chatila S. Fleury M. Ghallab and F. Ingrand. An architecture for autonomy. *The International Journal of Robotics Research*, 17:315–337, April 1998. 3.4

[46] A. Girard and G.J. Pappas. Hierarchical control system design using approximate simulation. *Automatica*, 45(2):566–571, 2009. 2.2

[47] S. Goldberg, M. Maimone, , and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *Proceedings of the IEEE Aerospace Conference*, pages 2025–2036, 2002. 5, 5.2

[48] C. Green and A. Kelly. Toward optimal sampling in the space of paths. In *Proceedings of the International Symposium of Robotics Research*, Hiroshima, Japan, November 2007. 4.2, 4.2, 5, 5.1, 6.3, 6.7, 7.1.1

[49] P. D. Grünwald and A. P. Dawid. Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory. *The Annals of Statistics*, 32(4), 2004. 6.6

[50] J. Guitton and J.-L. Farges. Taking into account geometric constraints for task-oriented motion planning. In *Proceedings of the Workshop on Bridging the Gap Between Task and Motion Planning at the International Conference on Automated Planning and Scheduling*, 2009. 3.4

[51] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2): 100–107, July 1968. 2.5

[52] K. Hauser and J.-C. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Proceedings of the ICAPS09 Workshop on Bridging the Gap between Task and Motion Planning*, 2009. 3.4

[53] J. Henrikson. Completeness and total boundedness of the Hausdorff metric. *MIT Undergraduate Journal of Mathematics*, 1, 1999. 7.1.2

[54] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in prm planners. In *Proceedings of the International Conference on Robotics and Automation*, pages 1408–1413, April 2000. 2.5

[55] T. Howard. *Adaptive Model-Predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2009. 5.5.3

[56] T. Howard. *Adaptive Model-Predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2009. 8

[57] T. Howard, C. Green, D. Ferguson, and A. Kelly. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, 25(1):325–345, June 2008. 2.2, 5.5.1

[58] T. M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, February 2007. 2.4

[59] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, March 2002. 5.5.2

[60] D. Hsu, T. Jiang, J. Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of the Internatiooal Conference on Robotics and Automation*, Taipei, Taiwnn, September 2003. 6.2

[61] D. Hsu, G. Sánchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proceedings of the International Conference on Robotics and Automation*, pages 3885–3891, 2005. 6.2

[62] D. Hsu, J.C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Intl. Journal of Robotics Research*, 25(7):627–643, 2006. 6.2

[63] W. K. Hyun and I. H. Suh. A hierarchical collision-free path planning algorithm for robotics. In *Proceedings of the International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, USA, August 1995. 2.2

[64] L. Jaillet and T. Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotics Research*, 27(11–12):1175–1188, 2008. 7

[65] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2005. 6.2

[66] A. Jain and C. C. Kemp. Pulling open doors and drawers: Coordinating an omni-directional base and a compliant arm with equilibrium point control. In *Proceedings of the International Conference on Robotics and Automation*, pages 1807–1814, May 2010. 3.4

[67] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4): 620–630, May 1957. 6.6

[68] L. P. Kaelbling and T. Lozano-Perez. Hierarchical planning in the now. In *Proceedings of the International Conference on Robotics and Automation*, Shanghai, China, May 2011. 3.4

[69] M. Kalisiak and M. van de Panne. RRT-blossom: RRT with a local flood-fill behavior. In *Proceedings of the International Conference on Robotics and Automation*, Orlando, Florida, USA, May 2006. 2.4, 10

[70] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011. 4.2, 4.2, 4.5,

5.5.2

[71] D. Katz, J. Kenney, and O. Brock. How can robots succeed in unstructured environments? In *Proceedings of Robot Manipulation: Intelligence in Human Environments at Robotics: Science and Systems*, Zurich, Switzerland, June 2008. 3.4

[72] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *Proceedings of the International Conference on Robotics and Automation*, pages 566–580, 1996. 7, 7.2.1, 8

[73] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996. 5.5.2

[74] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601, July-August 2003. 2.4

[75] A. Kelly and A. Stentz. Rough terrain autonomous mobility - part 1: A theoretical analysis of requirements. *Autonomous Robots*, 5(2):129–161, May 1998. 5.5

[76] A. Kelly and A. Stentz. Rough terrain autonomous mobility - part 2: An active vision, predictive control. *Autonomous Robots*, 5(2):163–198, May 1998. 5, 5.5

[77] A. Kelly, A. Stentz, O. Amidi, M. W. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. M. Vallidis, and R. Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *International Journal of Robotics Research*, 25(1):449–483, May 2006. 2.2, 5

[78] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, St. Louis, USA, March 1985. 7, 8

[79] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986. 1.1, 2, 2.3, 5.5.3

[80] R. A. Knepper, S. S. Srinivasa, and M. T. Mason. Curvature bounds on the weighted Voronoi diagram of two proximal paths with shape constraints. Technical Report CMU-RI-TR-10-25, Robotics Institute, Carnegie Mellon University, 2010. 1, 7.1

[81] R. A. Knepper, S. S. Srinivasa, and M. T. Mason. An equivalence relation for local path sets. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, Singapore, December 2010. 7

[82] R.A. Knepper and M.T. Mason. Path diversity is only part of the problem. In *Proceedings of the International Conference on Robotics and Automation*, Kobe, Japan, May 2009. 2.2, 4.4, 5.3

[83] R.A. Knepper and M.T. Mason. Empirical sampling of path sets for local area motion planning. In *Proceedings of the International Symposium of Experimental Robotics*, Athens, Greece, July 2008. 2.2, 3.2, 4.4, 5.3

[84] R.A. Knepper, S.S. Srinivasa, and M.T. Mason. Hierarchical planning architectures for

mobile manipulation tasks in indoor environments. In *Proceedings of the International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010. 3, 9.5

[85] S. Koenig and M. Likhachev. D*lite. In *Proceedings of AAAI/IAAI*, pages 476–483, 2002. 2.5

[86] Y. Koga and J.-C. Latombe. On multi-arm manipulation planning. In *Proceedings of the International Conference on Robotics and Automation*, 1994. 3.4

[87] L. Kovar, M. Gleicher, and F Pighin. Motion graphs. *ACM Transactions on Graphics*, 21 (3):473–482, July 2002. 5.5

[88] A. Lacaze, Y. Moscovitz, N. DeClaris, and K. Murphy. Path planning for autonomous vehicles driving over rough terrain. In *Proceedings of the ISIC/CIRA/ISAS Joint Conference*, pages 50–55, 1998. 5.2

[89] A. Lacaze, K. Murphy, and M. DelGiorno. Autonomous mobility for the demo iii experimental unmanned vehicles. In *Proceedings of AUVSI*, Orlando, FL, USA, July 2002. 5.2

[90] M. Lau and J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2006. 5.5

[91] J. P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Proceedings of Intelligent Autonomous Systems, An International Conference*, Amsterdam, The Netherlands, December 1986. 7

[92] J.-P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *Proceedings of the International Workshop on Intelligent Robots and Systems*, pages 765–773, 3–6 July 1990. 2.2

[93] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001. 8

[94] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001. 7, 7.2.1

[95] S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23 (7/8):673–692, July/August 2004. 7.2.3, 4

[96] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378, 2001. 2, 5.5.2

[97] T.-Y. Li and J.-C. Latombe. On-line manipulation planning for two robot arms in a dynamic environment. *The International Journal of Robotics Research*, 16(2):144–167, April 1997. 3.4

[98] T. Lozano-Perez, J. J. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournassoud, and A. Lanusse. Handey: A robot system that recognizes, plans, and manipulates. In *Proceedings of the International Conference on Robotics and Automation*, volume 4, 1987. 2.2

[99] V. Lumelsky and A. Stepanov. Automaton moving admist unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987. 7

[100] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Proceedings of the International Conference on Robotics and Automation*, May 2010. 2.2

[101] D. S. Minhas, J. A. Engh, M. M. Fenske, , and C. N. Riviere. Modeling of needle steering via duty-cycled spinning. In *Proceedings of the Conference of the IEEE EMBS*, 2007. 9.6

[102] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, pages 361–376, Utrecht/Zeist, The Netherlands, July 2004. 6.2

[103] J. R. Munkres. *Topology*. Prentice Hall, Upper Saddle River, NJ, 2000. 4.2, 7, 7.1.2, 7.2.2

[104] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods*. Society for Industrial Mathematics, Philadelphia, 1992. 4.2

[105] C. L. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. In *Procedings of the International Conference on Intelligent Robots and Systems*, 2000. 3.4

[106] I. R. Nourbakhsh. Using abstraction to interleave planning and execution. In *Proceedings of the Third Biannual World Automation Congress*, 1998. 2.2

[107] K. Olin and D. Y. Tseng. Autonomous cross-country navigation. *IEEE Expert*, 6(4): 16–20, 1991. 2.2

[108] M. Pivtoraiko and A. Kelly. Differentially constrained motion replanning using state lattices with graduated fidelity. In *Proceedings of the International Conference on Intelligent Robots and Systems*, September 2008. 2.2

[109] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(1):308–333, March 2009. 4.3, 5.5.1, 5.5.3

[110] E. Plaku and G. D. Hager. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Proceedings of the International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010. 2.2

[111] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003. 2.4

[112] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *Proceedings of the International Conference on Robotics and Automation, Workshop on Open Source Robotics*, Kobe, Japan, May 2009. 3.2

[113] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Proceedings of the International Conference on Robotics and Automation*, pages 802–807, Atlanta, USA, May 1993. 8

[114] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Pro-*

ceedings of the International Conference on Robotics and Automation, volume 2, pages 802–807, Atlanta, GA, 1993. 2.3, 5.5.3

[115] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of the International Conference on Robotics and Automation*, May 2009. 2.3, 3.3.3, 5.5.3, 8

[116] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, J. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier, J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Glke, F. Klose, and B. Rumpe. Caroline: An autonomously driving vehicle for urban environments. *Journal of Field Robotics*, 25(9):674–724, 2008. 5

[117] S. Ravi, D. Rosenkrantz, and G. Tayi. Facility dispersion problems: Heuristics and special cases. *Algorithms and Data Structures*, pages 355–366, 1991. 4.2, 5.1

[118] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990. 2.2, 2.5, 5.5

[119] J. Reif and H. Wang. The complexity of the two dimensional curvature-constrained shortest-path problem. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics*, pages 49–57, June 1998. 2.1

[120] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character animation. *ACM Trans. Graph.*, 26(4), October 2007. 5.5

[121] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5), October 1992. 8

[122] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5: 115–135, 1974. 2.2

[123] P. Sampl. Medial axis construction in three dimensions and its application to mesh generation. *Engineering with Computers*, 17(3):234–248, 2001. 1

[124] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proceedings of the International Symposium on Robotics Research*, Lorne, Victoria, Australia, November 2001. 5.5.2, 8

[125] G. Sánchez and J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *International Journal of Robotics Research*, 21(1): 5–26, 2002. 7, 7.1.3

[126] C. Schlegel. Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. In *Proceedings of the International Conference on Intelligent Robots and Systems*, Victoria, BC, Canada, October 1998. 2.4

[127] E. Schmitzberger, J.L. Bouchet, M. Dufaut, D. Wolf, and R. Husson. Capture of homotopy classes with probabilistic road map. In *Proceedings of the International Conference on Intelligent Robots and Systems*, October 2002. 7

[128] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23:729–746, 2004. 3.4

[129] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 1996. 5, 7

[130] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the International Conference on Robotics and Automation*, pages 3310–3317, May 1994. 2.5

[131] I. A. Suçan and L. E. Kavraki. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *Proceedings of the International Conference on Robotics and Automation*, Shanghai, China, 2011. 3.4

[132] K. Sun and V. J. Lumelsky. Motion planning for three-link robot arm manipulators operating in an unknown three-dimensional environment. In *Proceedings of the Conference on Decision and Control*, 1991. 9.5

[133] M. Tesch, K. Lipkin, I. Brown, R. Hatton, A. Peck, J. Rembisz, and H. Choset. Parameterized and scripted gaits for modular snake robots. *Advanced Robotics*, 23:1131–1158, 2009. 3.5.1

[134] C. E. Thorpe. Path relaxation: Path planning for a mobile robot. In *Proceedings of the National Conference on Artificial Intelligence*, pages 318–321, 1984. 8.3

[135] A. F. van der Stappen V. Boor, M. H. Overmars. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of the International Conference on Robotics and Automation*, pages 1018–1023, 1999. 6.2

[136] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, Guanajuato, Mexico, 2008. 3.4

[137] J. van den Berg, S. Patil, R. Alterovitz, P. Abbeel, and K. Goldberg. Lqg-based planning, sensing, and control of steerable needles. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, Singapore, December 2010. 9.6

[138] M. Vendittelli, J. P. Laumond, and C. Nissoux. Obstacle distance for car-like robots. *IEEE Transactions on Robotics and Automation*, 15:678–691, 1999. 2

[139] B. Vidakovic. *Γ-minimax: A paradigm for conservative robust Bayesians*, volume 152 of *Lecture Notes in Statistics*, pages 241–259. Springer-Verlag, New York, 2000. 6.6

[140] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H. Wuensche. Driving with tentacles: Integral structures for sensing and motion. *Journal of Field Robotics*, 25(9):640–673, 2008. 5

[141] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, and T. Kanade. First results in robot road-following. Technical Report CMU-RI-TR-86-4, Robotics Institute, Carnegie Mellon University, 1986. 2.2

[142] L.C. Wang, L.S. Yong, and M.H. Ang. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. In *Proceedings of the International Symposium on Intelligent Control*, Vancouver, BC, Canada, October 2002. 2.2

[143] R. J. Webster III, J. S. Kim, N. J. Cowan, G. S. Chirikjian, and A. M. Okamura. Nonholonomic modeling of needle steering. *International Journal of Robotics Research*, 25(5–6): 509–525, 2006. 9.6

[144] S. A. Wilmarth, N. M. Amato, and P. E. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings of the International Confemce on Robotics and Automation*, Detroit, Mich., USA, May 1999. 2.5

[145] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2010. 3.4

[146] Y. Yang and O. Brock. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In *Proceedings of Robotics: Science and Systems*, Philadelphia, PA, USA, August 2006. 2.2

[147] Y. Yang and O. Brock. Adapting the sampling distribution in prm planners based on an approximated medial axis. In *Proceedings of the International Conference on Robotics and Automation*, New Orleans, La., USA, April 2004. 2.5

[148] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987. 1

[149] Y. Yu and K Gupta. An information theoretic approach to view point planning for motion planning of eye-in-hand systems. In *Proceedings of the International Symposium on Robotics Research*, pages 306–311, 2000. 6.2, 7

[150] B.D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J.A. Bagnell, M. Hebert, A.K. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *International Conference on Intelligent Robots and Systems*, pages 3931–3936, October 2009. 3.5.3