Planning Under Uncertainty in Resource-Constrained Systems

by

Daniel DeWitt Strawser

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Signature redacted

Author ....
Department of Mechanical Engineering
August 9, 2019

Signature redacted

Certified by ....
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Certified by ....
John J. Leonard
Professor of Mechanical Engineering
Thesis Committee Chairman

Signature redacted

Accepted by ....
Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Planning Under Uncertainty in Resource-Constrained Systems

by

Daniel DeWitt Strawser

Submitted to the Department of Mechanical Engineering
on August 9, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

As autonomous systems become integrated into the real world, planning under uncertainty is a critical task. The real world is incredibly complex and systems must reason about factors such as uncertainty in their movements, environments, and human behavior. In the face of this uncertainty, agents must compute control trajectories and policies that enable them to maximize their expected performance while respecting probabilities on mission failure. The task is difficult because systems must reason about large numbers of scenarios concerning what may happen. Compounding the difficulty, many systems must reason about uncertainty while being resource-constrained. Autonomous cars and robots are time-limited because they must react quickly to their environment. Applications such as smart grids are computationally-limited because they require low-cost hardware in order to keep energy cheap.

Because of its importance to real world applications, a large amount of work has been devoted to planning under uncertainty. However, few applications focus on the resource-limited case. In time-constrained applications such as motion planning under uncertainty, methods typically focus on simplistic cost functions and models of the environment, dynamics, and stochasticity. In computation-constrained applications such as resource allocation for smart grids, approaches require computationally-intensive solvers that are unsuitable when system cost must be kept to a minimum. In both cases, the prior art is inadequate for real world demands.

In this thesis, I develop a series of algorithms that enable resource-constrained systems to better plan under uncertainty. First, my work enables time-constrained systems to better approximate expected cost, search in non-convex regions, and reason about more complicated environmental geometries and agent dynamics. Additionally, I propose algorithms that enable resource allocation under uncertainty on ultra low-cost hardware by distributing computation and approximating state uncertainty through a discretization.

In the time-constrained case, I model the problem of motion planning under uncertainty as a hybrid search that consists of an upper level region planner and a lower level motion planner. First, I present a method for quickly computing expected path cost and that is able to vary the precision with which the stochastic model is evaluated as required by the search. Next, I present the Fast Obstacle eXploration (FOX) algorithm that quickly generates a constraint graph for the region planner from complex obstacle geometries. I present a method for integrating CDF-based chance constraints with FOX as well as a method to model the collision probability of agents with non-trivial geometry. The latter algorithm transforms a complex problem in numerical integration into a simpler problem in computational geometry; importantly, the algorithm allows for massive parallelization on GPUs. Finally, this part concludes with the Shooting Method Monte Carlo (SMMC) algorithm. This algorithm models the chance constraint of dynamical systems with non-Gaussian state uncertainty. SMMC combines a shooting trajectory optimization with Monte Carlo simulation to approximate the collision probability for nonlinear dynamical systems.

In the computation-constrained case, I develop a set of resource-allocation algorithms that are able to reason about uncertainty while being implementable on ultra low-cost hardware. A market for reliability algorithm is presented that solves resource allocation under uncertainty by allowing a distributed set of agents to bid for reliability. A centralized planner is also presented where the resource allocation problem is modeled as a Markov Decision Process and a stochastic local search used to compute good policies.

The approaches for resource-constrained planning under uncertainty are benchmarked against a variety of test cases. In the motion planning domain, test cases are presented using linear dynamical systems, a Dubins car model, and a Slocum glider AUV. The method for bounding expected cost is shown to perform well against a sampling-based approach. The approach to dealing with complex obstacle geometry is shown to reach better solutions more quickly than a sampling-based RRT approach and disjunctive linear program. The sampling-based collocation method is shown to better approximate trajectory risk in simulation than a CDF-based model. Likewise, Shooting Method Monte Carlo is shown to better approximate trajectory risk than a sampling-based collocation method. In both sampling-based cases, GPU parallelization is shown to help scale computation of the chance constraint to large numbers of samples versus CPU-based computation. In the resource allocation case, the distributed and centralized contingency planners are benchmarked against mixed integer linear programs (MILP) and are shown to achieve comparable performance with a much smaller computational footprint.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Thesis Committee Chairman: John J. Leonard
Title: Professor of Mechanical Engineering

Acknowledgments

While many years contributed to the writing of this thesis, the following two pages are possibly the hardest to write. The challenge comes not from deriving equations or implementing an algorithm but from condensing the people, lessons, and memories into such a short space. Completing this thesis has been my most challenging undertaking up to this point in my life. Amid the seemingly endless nights in lab, compilation errors, worked weekends, changes in research directions, there has been a stream of people who have carried me forward over the years and prevented me from giving up.

First, I would like to thank the group members of the MERS group at MIT. Many of the ideas in this thesis result from ideas discussed with lab members or inspired by lab members' previous work. To name a few of the members during my tenure: Simon, Eric, Andrew, Pedro, Steve, James, Peng, Larry, David, Enrique, Marlyse, Ameya, Nick, Szymon, Jonathan, Ben, Zach, Sylvie, Nikhil, Meng, Yuening, Nan, Sungkweon, Jingkai, Cyrus, Sang, and Suleeporn. I would like to thank those who provided me with mentorship over the years such as the post-docs and research scientists in our lab: Andreas, Tiago, Erez, Christian, and Ashkan. I would like to thank Rich Camilli for his insights into the challenges of undersea exploration. My committee members John and Warren were also an invaluable resource for providing technical expertise and encouragement. Finally, I would like to thank my advisor, Brian Williams, for providing me with this opportunity. Brian was always accepting when things did not go according to plan. The greatest motivation you can have to accomplish a task or piece of work is to love what you do, and Brian certainly inspired that.

I would also like to thank the Tata Program at MIT, which supported me for the first three years of my studies. While I switched directions over the second half of my PhD to focus on motion planning under uncertainty, the insights and experiences that I gained during this program are invaluable. Specifically, I would like to mention some teammates of the uLink project, including Wardah, Reja, Rajeev, Rob, Szymon, Varun, Ahmet, Erik, and Victor. I will never forget the challenges and excitement of attempting to bring an important technology to market in a faraway land.

I would also like to thank the East Campus community at MIT where I served as a Graduate Resident Tutor (GRT) for my time in the PhD program. I spent the majority of my years as a PhD student on Second West (PTZ). The memories created on PTZ will remain special from roasting a lamb every Thanksgiving to preparing finals breakfasts.

Of course, this thesis would not be possible without the support of family. I would like to thank my parents, Ann and Carson, for their support over the (many) years of the PhD and trying to avoid questioning why I was putting off getting a real job for so long. Also important to mention are my brother and sister, Matthew and Amanda, and my numerous nieces and nephews. Finally, I thank my finacée Mary and my dog Henry Charoo. Without the two of you, I would have given up this thesis many years ago.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As autonomous systems become more prevalent outside of the lab and are placed in real-worlda circumstances, they must make decisions in increasingly complex situations. Autonomous cars must deal with complex traffic patterns while quickly navigating city streets. Underwater autonomous vehicles (AUVs) must consider uncertain ocean currents while accomplishing scientific missions and navigating the ocean floor. Smart energy management systems are proliferating due to low cost microcontrollers and device connectivity. The surge in cheap solar and wind power means that renewable energy is finally becoming competitive with fossil fuels, but the variability in renewables can destabilize the grid. In order to handle this variability and make better resource allocation decision, smart energy systems and the electricity grid itself must also make decisions under uncertainty. All of these systems share the characteristic of needing to reason about uncertainty while being resource-constrained. Autonomous cars and drones are time-constrained: they must make decisions quickly in uncertain, complex environments. Smart energy systems are typically cost-constrained and must reason about uncertain energy generation and demand on low cost, embedded systems.

1.1 Time-Constrained Planning Under Uncertainty: Autonomous Agents

As they become more widespread in the real-world, one of the toughest challenges faced by autonomous systems such as cars, robots, and AUVs is determining how to act in the face of complex uncertainty. An autonomous system has a set of goals to accomplish and it must determine an appropriate control policy to complete these goals while respecting system constraints. In most systems that operate in unconstrained environments, computing this control policy is a very difficult task due to uncertainty.

There are many types of uncertainty that an agent may experience. First, an agent must reason about uncertainty about itself. It may not know exactly where it is in the world. There may be uncertainty in what will happen or where exactly it will go when it takes an action; consider an autonomous car deciding whether to speed up when driving on a slick road. It is possible the increased acceleration will cause a loss of control. Second, an agent may be uncertain about its environment. For example, an autonomous underwater vehicle (AUV) has the common task of exploring the ocean floor for interesting features. The AUV likely has limited information about underwater obstacles and large amount of uncertainty about ocean currents that may rapidly push it into an obstacle. Finally, autonomous agents must reason about uncertainty in other agents and humans. A manufacturing robot working with humans must reason about the humans' intentions and likely actions when deciding what to do.

Planning under uncertainty in autonomous systems is important but very hard. The hardness comes from the fact that the most general problems are intractable; in order for autonomous systems to accurately plan they must reason over a huge number of possible scenarios or events that may occur. Compounding the problem is the fact that these agents must plan quickly. Many autonomous systems that deal with real-world environments must plan and react in seconds or milliseconds. This means that planning under uncertainty requires autonomous systems to reason about large numbers of scenarios very quickly.

Part 1 of this thesis focuses on uncertainty in path and motion planning. This refers to the problem of an agent navigating through its environment and avoiding obstacles while uncertainty exists in the agent's state or position. This problem can be formulated in terms of a chance-constrained optimal control problem. This is a strategy for computing a trajectory that optimizes some objective function such as the time traveled while respecting a probability of mission failure, such as limiting the probability of obstacle collision. Although prior work has applied these methods, most approaches have focused on agents with linear dynamics navigating simple environments. The goal of this part of this thesis is to extend these approaches to agents with more expressive dynamics and geometries navigating complex environments.

1.1.1 Requirements for the Planner

In order to be applicable to real-world systems, I design my approch with a number of requirements.

Accounting for Stochasticity

The approach must be able to reason about uncertainty when computing a trajectory. This means that the system must be able to minimize an expected cost function while respecting a probabilistic or chance constraint. My approach focuses on uncertainty in the agent's state.

Realistic Environment Geometry

Many approaches for path & motion planning under uncertainty focus on simplistic 2D rectangles and triangles. An approach that seeks to model real-world autonomous systems should be able to model reasonable approximations to three dimensional environment geometry that is not necessarily convex.

Complex Dynamical Systems & Agent Geometries

The approach should be able to compute trajectories for complex dynamical systems with non-trivial geometries. Many approaches focus on linear time invariant (LTI) systems that model robots without geometry (i.e. point robots). Real-world systems are often nonlinear and always have geometric dimensions that must be considered when navigating environments.

Speed

In order for an autonomous system to interact with a complex and ever-changing environment, it must be able to plan quickly. For this reason, many autonomous systems are time-constrained with respect to planning. The approach should be fast enough to be potentially applied online and allow the agent to replan.

1.1.2 Components of My Approach

To satisfy the requirements presented above, there are a few key components of my approach.

Hybrid Search

The overall approach to the optimal control problem is a hybrid search strategy. The general idea is that an upper level graph search algorithm, the region planner, is used with a lower level continuous state optimization subroutine, the trajectory planner. This idea has been applied before for combined activity and motion planning, albeit in the deterministic case [34].

Intuitively, the idea rests on the fact that certain optimization procedures are better suited to solving different types of problems. Linear and nonlinear mathematical programs have seen success solving trajectory optimization problems where the decision variables are continuous state variables and control inputs. In many cases, these types of programs are inadequate for solving a planning problem such as dealing with multiple goals or a non-convex obstacle-free region due to the presence of obstacles.

In these cases, some type of discrete decision variable is typically required to encode which activity to take next or on which side of an obstacle an agent should travel. These discrete decision variables can be modeled as nodes in a graph.

Approximating the Expected Path Cost

For motion and task planning under uncertainty, agents must reason about the *expected* cost or utility of accomplishing a task rather than the deterministic cost. In general, the expected cost is written as an expected value of an agent's state, control inputs, and time and is generally intractable to compute. Because of the difficulty of computing an agent's expected utility function, it is typically approximated as a deterministic function or a utility simple function is assumed for which an analytic solution to the expectation is known.

I propose an approach that provides a numerical bound on an agent's expected cost function in the case of convex cost functions. An advantage of this bound is that the computational effort can be scaled as needed. Typically, approaches to planning under uncertainty derive a simplified model of uncertainty and then compute a policy with respect to the simplified model. My approach integrates the precision of the stochastic model with the search.

Complicated Environment Geometry

To handle complex environmental geometry, I propose an approach named Fast Obstacle eXploration (FOX). FOX models environments using polytopes and enables the generation of geometric constraints directly from the environment geometry. The geometric constraints are termed regions in this thesis; regions that help an agent avoid obstacles are landmark regions. An environment of regions can be viewed as forming a graph, a region graph. Nodes in the graph indicate geometric constraints on the agent's trajectory. Sequences of regions can be passed to a trajectory optimization subroutine. The insight in building the graph is that environments contain large amounts of geometric information; only a small number of surfaces are likely to be active in motion planning problems. An algorithm should attempt to only generate relevant geometric constraints when computing paths.

Handling Agents with Nontrivial Geometry

While many approaches from literature focus only on risk-aware planning for point robots, real-world autonomous systems have nontrivial geometric proportions that must be considered when computing collision risk. Computing the risk of collision for these systems is very challenging because of the complex integration that is required. My insight is to turn the challenging integration into a more tractable collision checking problem and use sampling to approximate the risk of collision. This process can take advantage of parallelization on GPUs.

1.1.3 Shooting Method Monte Carlo

Many autonomous systems of interest are described by nonlinear differential equations. The nonlinearity makes planning under uncertainty very difficult because the resulting probability distribution describing the agent's state is complex. While past approaches make assumptions including linearizing the system dynamics and assuming only Gaussian uncertainty, I propose an approach that combines the shooting method of trajectory optimization with Monte Carlo sampling. This enables me to reason about complex probability distributions for nonlinear systems.

1.1.4 Input & Output

My planner takes as input an agent whose evolution is described by a set of dynamical equations. The dynamical equations also include a probabilistic model that describes the uncertainty in the agent's movement. Similar to other motion planning formulations, the agent must navigate from an initial state to a goal state. It must avoid obstacles in its environment, which are represented by polytopes. The input can be represented as a chance constrained qualitative state plan (CCQSP) as proposed in [80]. The planner's output is a deterministic, open loop control trajectory that enables the agent to accomplish its goals while respecting a maximum bound on the probability of mission failure.

1.2 Resource-Constrained Planning Under Uncertainty: Smart Energy Systems

One of the world's most pressing issues is the lack of reliable electricity. Over 1.3 billion people in the world lack reliable electricity, which results in lower education, healthcare, and economic growth [1]. Many of these people are located in remote geographies that make grid extension impractical [75]. Locations with grid connection often experience unpredictable supply due to electricity theft and lack of generation; approximately 700 million people in India live with unreliable electricity, many with only a few hours per week [24]. One popular solution is solar home systems. In Bangladesh, more than one million solar home systems have been adopted [93]. However, solar generation is uncertain and to provide adequate reliability solar home systems must oversize their batteries and therefore result in a higher cost per watt. One alternative is a microgrid, which is a self-contained electricity grid, often disconnected from the main grid. Microgrids have been proposed to provide cheaper and more reliable electricity because they can aggregate loads, storage, and generation. Through efficient scheduling algorithms, they can reduce peak power demand and provide a lower cost per watt [97].

The problem of electricity-scarcity in the developing world and the proposed microgrid solution motivates the second part of this thesis, which focuses on algorithms for resource allocation under uncertainty in computationally-constrained systems. This work was done as part of the uLink project, a project that sought to improve developing world electrification by using ad hoc microgrids. The uLink is a shoe box-sized device into which users could plug their loads and generators. The devices could be connected together and allow households to share electricity. The overall architecture is shown in Fig. 1-1.

Two features of the uLink device are important to this thesis. On a microgrid,

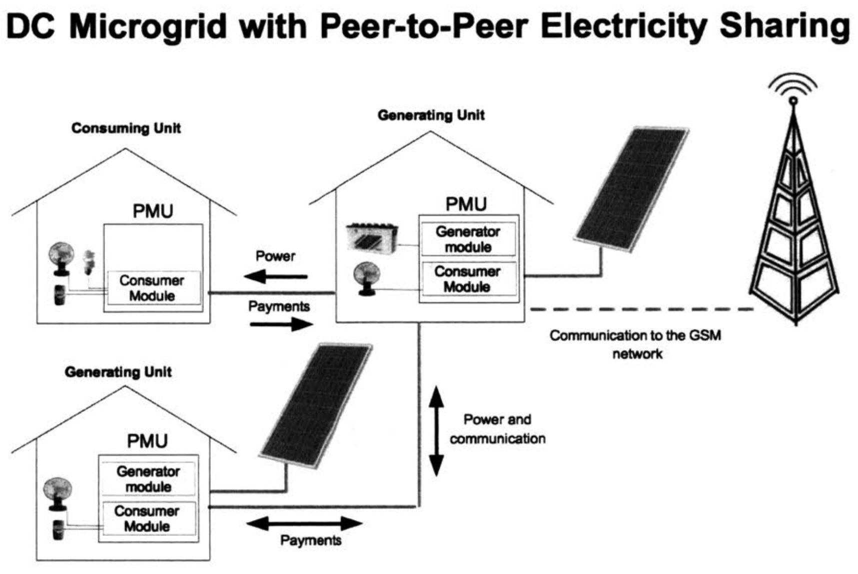


Figure 1-1: Concept of a microgrid architecture for the developing world, taken from [100], [49].

resource allocation under uncertainty is important to lowering electricity costs. Efficient resource allocation is necessary to ensure that critical loads receive adequate power. Without it, critical nighttime lighting loads would need to be curtailed or a larger (and more expensive) battery would need to be installed. Complicating the problem is the need to deal with uncertainty: both user electricity demand and solar generation serve as significant sources of uncertainty.

The second feature of the uLink project that is important to this thesis is the need to keep costs extremely low. In order to make an impact for the people who most need electricity, system hardware costs must be kept to a minimum and the algorithms implemented on \$2 - \$3 microcontrollers. These systems must perform resource allocation under uncertainty while being implemented on extremely computationally-limited devices.

Similar to the motion planning problem, there are a number of requirements for the resource allocation algorithm:

Take into Account Generation & Demand Uncertainty

The resource allocation problem for the electricity use case is challenging because both demand and generation are uncertain. It is uncertain when users will want electricity and how much they will want. Because the grid relies on solar generation, the amount of electricity generated is also not deterministic.

Handling Additional Constraints

Electricity scheduling poses additional constraints in addition to the capacity constraints posed by traditional resource allocation. For example, batteries are especially important to the microgrid because they allow it to shift load; without batteries none of the users would receive energy during the night. Batteries are difficult for the resource allocation algorithm because they allow the system to shift resource availability (up to the battery's capacity).

Feasible on Low Cost Hardware

As mentioned above, the microgird use case is characterized by an extremely low ability to pay. Therefore, the solution must also be extremely low-cost. This translates into the need to deploy the system on extremely resource and memory-limited microcontrollers. This is challenging for resource allocation under uncertainty because such systems must typically reason about complex probability distributions, which requires a large amount of memory.

This thesis presents two approaches toward satisfying these requirements: a distributed market-based approach and a centralized MDP-based approach.

1.2.1 Distributed Market-Based Approach

A distributed approach is presented that uses a novel tatonnement algorithm where agents make bids on electrical activities they would like to receive as well as their quality of service. A central market clears the bids and schedules the agents in time and reliability. The approach is memory-efficient because the market clears through a Monte Carlo simulation that approximates the probability of agents receiving power.

1.2.2 Centralized MDP-Based Approach

A second centralized scheduling approach is presented, which accounts for uncertainty in electricity generation and consumption. The uncertain state is modeled as a Markov Decision Process (MDP). A stochastic local search that incrementally improves an incumbent policy is able to compute good policies in extremely resource-limited environments.

1.3 Thesis Contributions

This thesis presents the following contributions, divided between the motion planning under uncertainty and the resource allocation under uncertainty use cases.

1. A hybrid search algorithm that transforms the non-convex path planning under uncertainty problem into a graph search and a continuous state optimization subroutine

2. A bounding approximation to the expected cost function that allows quick evaluation of path costs and the ability to quickly prune regions of the search space.

3. Fast Obstacle eXploration (FOX): An algorithm that transforms 2/3 dimensional polytope obstacles into constraints for use in the upper level graph search.

4. A method to compute risk for agents with non-trivial geometries that can be integrated into the hybrid search routine

5. Shooting Method Monte Carlo (SMMC): An approach that combines the shooting method from trajectory optimization with a Monte Carlo simulations that enables risk-aware trajectory planning for agents with nonlinear dynamics

6. A market-based algorithm that allows a distributed set of agents to reach consensus under conditions of uncertainty

7. A centralized contingent planner that models the resource state as a Markov Decision Process (MDP) and an efficient stochastic local search that enables implementation on low cost hardware

1.4 Thesis Organization

This thesis is divided into two parts, both of which focus on planning under uncertainty in resource-constrained systems. The first part focuses on motion planning under uncertainty. These systems are resource-constrained in time; to operate in the real-world, the systems must quickly reason about a large number of scenarios and output a control trajectory. Chapter 2 sets the stage by introducing prior work and the problem of motion planning under uncertainty. Next, the chapters focus on approaches outlined in the introduction. Chapter 3 explains a method for approximating expected cost. Chapter 4 introduces Fast Obstacle eXploration (FOX), a method for planning under uncertainty with complex obstacle geometries. Chapter 5 explains how collision risk (via chance constraints) can be integrated int FOX. Chapter 5 also provides test cases and numerical benchmarks of FOX against other approaches. Chapter 6 focuses on extending FOX to agents with nontrivial geometries. Chapter 7 explains Shooting Method Monte Carlo (SMMC), an algorithm for planning under uncertainty for agents with nonlinear dynamics.

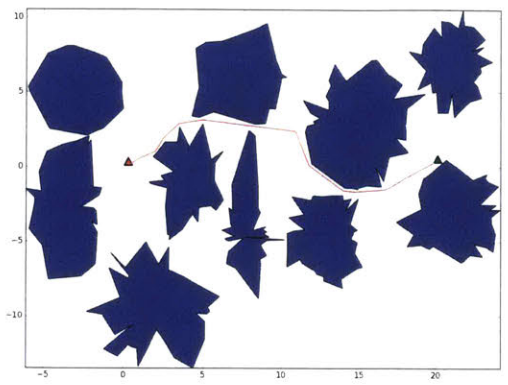
The second part of this thesis focuses on resource allocation under uncertainty. The resource allocation algorithms focus on developing world microgrids; the systems are resource constrained because the application calls for ultra low-cost platforms. This is described more fully and the problem introduced in Chapter 8. Chapters 9 and 10 explain approaches to resource allocation on such low cost systems through a distributed market-based approach and centralized MDP-based approach.

Chapter 2

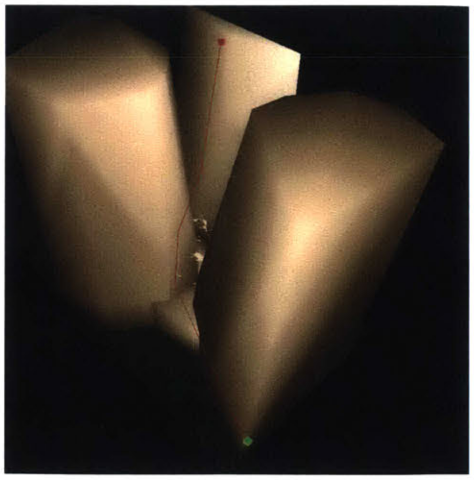
Motion Planning Under Uncertainty

Motion planning under uncertainty is relevant to most autonomous systems that operate in the real world. Systems such as autonomous underwater vehicles (AUVs) and autonomous cars must compute control policies in the face of dynamical and environmental uncertainty. A common task of these systems is traveling from an initial location to a goal location while maintaining a bound on the risk of colliding with an obstacle. Performing this risk-aware planning task efficiently while dealing with complex objectives, non-trivial obstacle & agent geometries, and nonlinear agent dynamics is the focus of this part of the thesis, Chapters 2 - 8.

The problem is difficult because the general form of risk-aware motion planning is intractable. Computing the probability of obstacle collision would require an agent to integrate over complex probability distributions quickly. Much progress has been made in simplifying the stochastic model and formulating the motion planning problem as an optimal control problem [80], [14]. However, most optimal control approaches focus on efficient approximations to stochasticity and do not consider obstacle geometries beyond 2D convex polygons (many consider only 2D triangles or rectangles). Non-trivial geometry, such as might be encountered in the real world, poses two challenges: modeling the probability of failure (often called a *chance constraint*) with respect to this geometry and efficiently finding a good path. Optimal control approaches often rely on convex optimizers or mixed integer linear programs (MILPs) which either cannot encode complex geometry or are inefficient at dealing



(a) Planning in 2D around complex shapes.



(b) Planning in 3D around the Hawaiian islands.

Figure 2-1: A key task of autonomous systems is to travel from an initial location to an end location while avoiding obstacles. A good planner should model uncertainties in the movement & environment to allow for a margin of safety when the plan is executed.

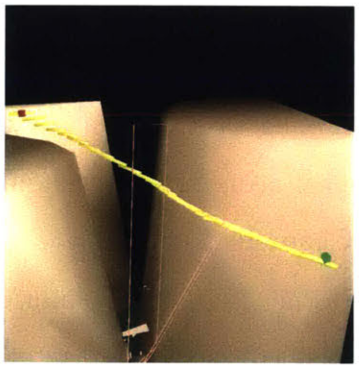


Figure 2-2: Motion planning algorithms should produce paths that are dynamically feasible, meaning they respect the agent's dynamical equations of motion such as in this example featuring a Slocum glider AUV. This is very challenging because equations of motion can be complex; oftentimes finding even a feasible trajectory is non-trivial.

with complex environments.

2.1 Problem Definition

I consider the problem of an agent navigating from an initial state $\mathcal{I}$ to a goal state $\mathcal{G}$ while avoiding obstacles with probability $1-\epsilon$. The purpose of the algorithm is to output an open loop control trajectory $\mathbf{u}_k \; \forall \; k \in K$. The control outputs $\mathbf{u}$ represent inputs to the agent's actuators. Throughout this thesis I use the term *agent* to denote a general autonomous system; it may represent an autonomous car, an AUV, a robot, or another system.

The navigation problem stated above can be framed as an optimal control problem as follows:

$$\begin{aligned}
\text{minimize} \quad & \mathbb{E}\left[f_0\left(\mathbf{x}_{0:K}, \mathbf{u}_{0:K-1}\right)\right] && (2.1)\\
\text{subject to} \quad & \dot{\mathbf{x}} = f\left(\mathbf{x}, \mathbf{u}\right) + g\left(\mathbf{w}\right) && (2.2)\\
& P\left(\bigwedge_{k \in K} \mathbf{x}_k \in \mathcal{W}_{free}\right) \geq 1 - \epsilon && (2.3)\\
& \mathbf{x}_0 \in \mathcal{I} && (2.4)\\
& \mathbf{x}_K \in \mathcal{G} && (2.5)
\end{aligned}$$

Term (2.1) is the objective function; it requires minimization of the expected trajectory cost. Minimizing this term ensures that the output policy is *good* in some way. There are a wide number of forms the objective can take from minimizing mission time to minimizing fuel consumption or path distance. Importantly, because of system stochasticity, it is written as an expected value. It is not possible to guarantee what the cost of a trajectory will be (the deterministic cost) but the algorithm must produce the best output given system uncertainty.

Constraint (2.2) represents the agent's dynamics or equations of motion. It is written generally because there are many possible numerical integration / discretization schemes when performing a trajectory optimization (e.g. Euler, trapezoidal). The left-hand side (LHS) is the time derivative of the agent's state $\mathbf{x}$. The right-hand side (RHS) consists of two terms. The term $f\left(\mathbf{x}, \mathbf{u}\right)$ represents the agent's deterministic equations of motion. The second term, $g\left(\mathbf{w}\right)$, represents the problem's source of uncertainty. Variable $\mathbf{w}$ is a vector of random variables. A simple case is additive Gaussian noise where $g\left(\mathbf{w}\right) \sim \mathcal{N}\left(0, \Sigma\right)$, i.e. $\mathbf{w}$ are *i.i.d.* multivariate Gaussians.

Constraint (2.3) is the probabilistic or chance constraint which ensures that the probability of obstacle collision is at most ϵ. The constraint is challenging because of the need to integrate the state probability distribution over $\mathcal{W}_{free}$, the obstacle free region of the workspace space. The integral is often intractable because of the need to

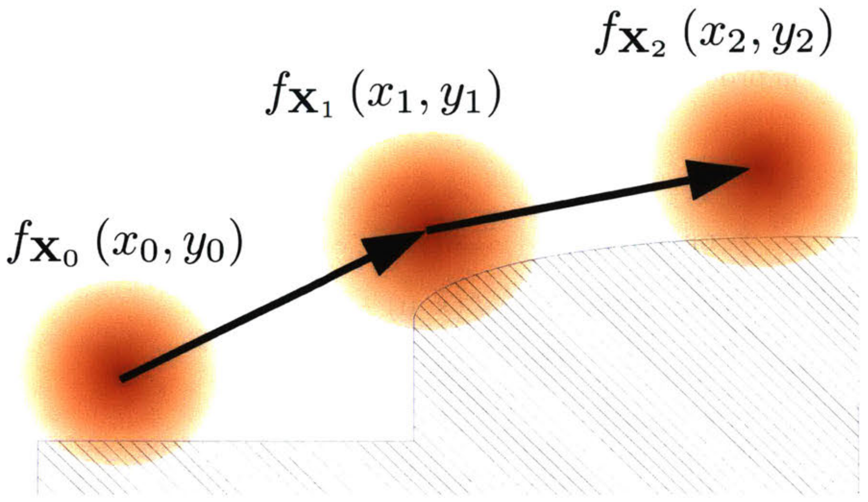

$f_{\mathbf{X}_0}(x_0, y_0)$
$f_{\mathbf{X}_1}(x_1, y_1)$
$f_{\mathbf{X}_2}(x_2, y_2)$

Figure 2-3: Even in a simple discrete-time, 2D case, motion planning under uncertainty is difficult. Consider the need to integrate the state distributions with respect to arbitrary obstacle geometry. This is made more difficult if the distribution $f_{\mathbf{x}}$ is arbitrary.

integrate complete probability distributions with respect to complex and non-convex geometric boundaries.

Terms (2.4) and (2.5) indicate that the agent starts in the region $\mathcal{I}$ and must end in the region $\mathcal{G}$. These regions can take different forms depending on the problem. Examples include requiring the robot to end within a polytope region (i.e. inside a 2D polygon or 3D polyhedron) or the stricter requirement of the goal being a single point. While I introduce the problem in terms of a single goal, the methods of this thesis can be extended to cases when the agent must consider a set of goals, $\boldsymbol{\mathcal{G}}$. This is explored more fully in Chapter 5, where the planner is used to compute trajectories for a glider exploring an undersea volcano.

2.2 Challenges

The general case of motion planning under uncertainty is very hard to solve. Even without uncertainty, path planning is PSPACE-complete [89]. Adding uncertainty means the agent must consider a very large (possibly infinite) number of scenarios; this makes the general problem intractable. There are a number of reasons why the problem is hard. The first two relate to deterministic versions of the problem and the final three focus on the stochastic version. An illustration of some of these issues are provided in Fig. 2-3.

2.2.1 Obstacles make the workspace non-convex

Even for simple 2D environments, the presence of obstacles makes finding a solution hard. Intuitively, this is because obstacles cause the workspace to be non-convex. Even a seemingly simple version of the problem where rectangular agents must move from initial positions to final positions (the "warehouse-man's problem") has been shown to be PSPACE-complete [44].

2.2.2 Complex dynamics

Motion planning requires an agent to move from an initial state to a goal state while respecting equations of motion that serve as constraints. In the case of nonlinear dynamics, these equations can be complicated and even finding an initial feasible solution is difficult (as opposed to an optimal one).

2.2.3 Complex distributions of state

A principle task for motion planning under uncertainty is modeling the distribution of an agent's state at some future time. This is important because it allows the agent to compute quantities such as the probability of collision, the expected cost of a trajectory, and other probabilities related to achieving its goal. In the general case, computing or propagating the agent's state distribution is a very difficult task for a few reasons.

First, real world agents are inherently continuous time systems. Their equations of motions are defined by a set of differential equations with time derivatives. Unfortunately, modeling continuous time stochasticity is very difficult. One must resort to a stochastic differential equation (SDE) for which the rules of calculus are different. Integrating these equations analytically is only possible in the simplest cases and numerical methods must be used.

If continuous time stochasticity is ignored and a discrete time stochastic process assumed, computing the agent's state distribution is still daunting. In only a small number of cases (such as additive Gaussian noise) can an agent's state distribution be calculated analytically. Even if the noise is additive Gaussian, nonlinear dynamics means that the agent's future state is not necessary Gaussian.

An additional modeling consideration is dependencies between multiple time steps. For example, in the discrete time case, if an agent is almost colliding with an obstacle at time t, it is likely the agent's controller will attempt to make an evasive maneuver to avoid the obstacle (i.e. slam on the brakes or apply a large control input in one direction). Because of uncertainties in the controller, this evasive maneuver will likely induce other uncertainties at later times. If this is the case, the model of uncertainty is correlated at various time steps. To avoid this complexity, a common approach is to assume some type of independence or mutual exclusivity between the time steps.

2.2.4 Integrating with respect to complex geometry

A typical approach to computing the probability that an agent collides with an obstacle is to integrate a probability distribution with respect to the free region of the workspace, which is defined by the obstacles' surface. Unfortunately, real world geometry can be incredibly complex. This means that it's necessary to integrate a complex function with respect to complex limits of integration. This integration must often be simplified by making assumptions regarding the obstacle geometry (i.e. 2D, convex, polygonal, etc.) or approximated numerically.

2.2.5 Computing the expected trajectory cost

The objective for the motion planning problem is in terms of an expected cost to reflect the fact that the state evolves stochastically. Similar to the computation of the collision probability, this requires a complex integration: a (possibly complex) cost function multiplied by a probability distribution. An approximation is often necessary where the designer uses simple cost function with an analytic solution, numerically approximates the expected cost, or ignores the expectation entirely and assumes that the trajectory has a deterministic cost.

2.3 Prior Work

In spite of the problem's complexity, its real world applicability has driven many advancements in recent years. A brief summary of select approaches and their relevance to this thesis are covered in the following section. Additional prior work is discussed in various chapters depending on the chapter's contribution. While path and motion planning is one of the most studied problems in the control of autonomous systems, this thesis focuses on risk-bound motion planning under uncertainty. For an excellent general reference on motion planning, the reader is referred to [69].

Much of the groundwork for planning under uncertainty was laid in the seminal work of [88]. There are a few general approaches to the risk-bound motion planning problem that have seen success in recent years: modeling the problem using discrete states, randomized sampling-based approaches, and trajectory optimization-based approaches.

A number of methods have modeled the problem of risk-aware planning using a discretization. Innovative approaches are presented in [96] and [95] where the authors perform risk-aware planning on partially observable Markov decision processes (POMDPs). Another work looks at using a POMDP to model the risk of collision in an unmanned aircraft with sensor uncertainty [104]. For these approaches to be successful, the continuous state must be discretized, which is often complicated.

A second set of approaches has used randomized sampling-based methods. Work in [74] extends the classical rapidly-exploring random tree (RRT) algorithm to chance-constrained instances. Another RRT-type planner is presented in [108] where the authors investigate uncertainty in motion planning when using a linear quadratic control law. Work in [48] builds a probabilistic roadmap and uses an efficient sampling routine and parallelization to approximate the chance constraint. A sampling-based approach in [14] models the collision risk and expected path cost by using an approach inspired by particle filters from state estimation. The resulting trajectory optimization requires a mixed integer linear program (MILP) because a binary indicator variable is required to indicate whether a sample collides with an obstacle.

A number of approaches have transformed the problem into one amenable to trajectory optimization. A focus has been on simplifying the general chance constraint into one that is amenable to mathematical programming. A simplification is presented in [13] where the authors approximate the joint chance constraint with a sum using Boole's inequality. This new constraint can be used in a disjunctive linear program (DLP) and is guaranteed to satisfy the original chance constraint. Other approaches by the same authors focus on simplifying the mathematical program into one amenable to convex optimization [15]. Because of efficient convex optimization routines, these approaches are able to produce solutions very quickly. Work in [79] presents Iterative Risk Allocation (IRA). IRA consists of a two-stage optimization procedure where the upper stage allocates risk for each constraint, while the lower stage optimizes the control input. Importantly, it is shown [79] that risk allocation is a convex mathematical program under certain conditions. Further refinements by the same authors in [80], present a non-convex version of iterative risk allocation that can deal with non-convex state spaces.

2.4 Approach - Hybrid Search

To solve the planning problem presented in Section 2.1, this thesis takes a hybrid search approach. The hybrid search consists of two levels: an upper level region planner and a lower level trajectory planner. Intuitively, the region planner determines which regions of the agent workspace should be explored as potential candidates for trajectories that travel from the initial state to the goal. Given an ordered set of regions, the lower level trajectory planner optimizes a trajectory that is constrained to travel through these regions. The approach is general; many methods are possible to model the regions, how the regions are connected together and searched, and how the

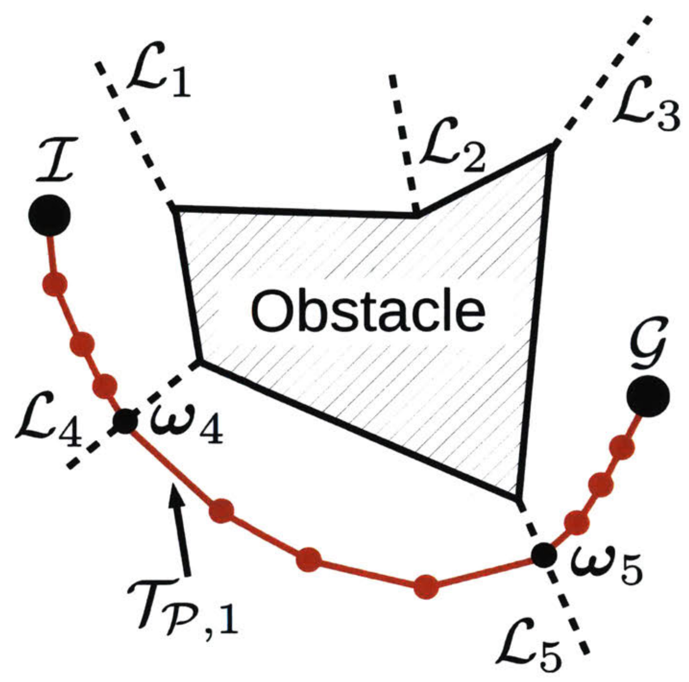


Figure 2-4: Example of the region planner and trajectory planner. Dotted lines represent waypoint constraints; ω_4 and ω_5 are waypoints which must lie on $\mathcal{L}_4$ and $\mathcal{L}_5$. Red dots represent knot points of the trajectory optimization where the trajectory is indicated as $\mathcal{T}_{\mathcal{P},1}$.

trajectory optimization is formulated and solved. Chapters 3 - 8 focus on developing strategies for each of these points. An overview of hybrid search is depicted in Fig. 2-4.

While different aspects of hybrid search are discussed throughout this article, a set of notation and terminology is standard. Specifically, the region planner's regions are denoted with $\mathcal{R}$. In general, $\mathcal{R}$ (and the term *region*) refers to some Euclidean space in the agent's workspace, $\mathcal{W}$. In this article, $\mathcal{R}$ has dimensionality less than or equal to d where d is the dimensionality of the workspace. For example, in a 2D environment, $\mathcal{R}$ can refer to a ray (1D) or a rectangle (2D) or a variety of other 2D shapes. It is possible that the dimensionality of $\mathcal{R}$ is greater than the workspace, i.e. if $\mathcal{R}$ also models constraints on the robot's configuration space. These methods are not considered in this work. Another assumption in this work is that $\mathcal{R}$ are convex. This allows them to be readily translated into trajectory constraints input to the trajectory planner.

One of the advantages of defining $\mathcal{R}$ to be general is to allow them to model a wide range of desired agent behaviors in the workspace. For example, a rectangle could represent a goal region. In Section 4.3.2, we introduce landmark regions, denoted by $\mathcal{L}$. Landmark regions are types of regions of dimensionality $d-1$, i.e. $\mathcal{L} \in \mathbb{R}^{d-1}$, which guide the agent around obstacles. Regions can also be combined to form motion primitives. Ordered sets of regions form paths, i.e. $\mathcal{P} = [\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_n, \ldots, \mathcal{R}_N]$. Regions along paths are indexed by n with the total number of regions being N. Intuitively, paths are sequences of regions that the region planner proposes as candidate sets of constraints to take the agent from some initial state to some goal state. It is unknown whether a feasible risk-bound trajectory exists that travels through $\mathcal{P}$ until the trajectory planner is complete. The region planner can be viewed as generating an undirected graph of $\mathcal{R}$; this graph is referred to as a *region graph.*

The region planner passes paths $\mathcal{P}$ to the trajectory planner, which then optimizes a trajectory $\mathcal{T}$ constrained to lie on $\mathcal{P}$ in some way. Trajectories $\mathcal{T}$ are sets of state and control variables that describe the agent's motion. The approaches in this thesis transcribe the optimal control problem into a trajectory optimization problem. This means that the functional form of the trajectories is determined by the integration scheme used by the trajectory optimizer. A common technique used in this thesis, although not the only one, is using the collocation method of trajectory optimization where trajectories are represented by an ordered set of knot points indexed by k, i.e. $\mathcal{T} = [\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_k, \mathbf{u}_k, \ldots, \mathbf{x}_K, \mathbf{u}_K]$. The trajectory is constrained to lie on the path $\mathcal{P}$ by constraining various knot points $[\mathbf{x}_k, \mathbf{u}_k]$ to lie on landmark regions $\mathcal{L}_n$. The constrained knot points are termed *waypoints.* They are depicted by $\boldsymbol{\omega}_4$ and $\boldsymbol{\omega}_5$ in Fig. 2-4.

Principally, hybrid search was selected to solve the motion planning problem because different search/optimization strategies are better at solving certain types of problems. As shown in prior work, continuous state trajectory optimization can be very good at solving for risk-bounded trajectories [80], [12]. Under certain conditions, solving the risk-bounded motion planning problem reduces to a convex program, for which many efficient algorithms exist to solve. Even if the trajectory optimization is non-convex due to more complicated nonlinear dynamical constraints, nonlinear optimization routines such as sequential quadratic programs (SQPs) can often solve the problem to local minima if the optimizers are provided *good* initial guesses for the decision variables. These approaches share the characteristic that the decision variables (i.e. trajectories) are continuous.

Difficulty enters when modeling the disjunctive constraints generated by obstacles. First, the presence of obstacles mean the workspace is non-convex and convex optimization does not apply without transforming the workspace in some way. Furthermore, in the presence of obstacles, it is difficult to derive good, feasible initial guesses for nonlinear optimization routines. On the other hand, the region planner provides an intuitive approach to dealing with the disjunctive constraint. The disjunction is modeled as a parent-child relationship in a search tree; traveling on one side of an obstacle corresponds to taking one branch in the tree and traveling on another side of the obstacle means taking another. In addition to dealing with the disjunction implied by the presence of obstacles, the region planner can incorporate other types of discrete choices such as multiple goals.

2.5 Assumptions

One of the hallmarks of motion planning under uncertainty are the assumptions that are required to make the problem tractable. Various assumptions are made in different chapters with regards to the environment's complexity, the complexity of the agent's dynamics and geometry, and the complexity of the probabilistic model. Certain assumptions present in one chapter are relaxed in another. For example, Chapter 3 deals with 2D environments with convex obstacles while subsequent chapters relax this assumption. Chapters 6, 7, 8 make progressively fewer assumptions about modeling trajectory risk.

A number of significant assumptions persist throughout the thesis. First, I assume that the environments are known a priori. This is a large assumption, particularly in environments where the agent must compute trajectories based on its incomplete knowledge of the world. However, in many circumstances, models of the environment are known such as in undersea use cases where bathymetry data is available or an agent navigating city streets with an accurate map. Even if the environment model is not precisely known, sensor-generated point clouds can readily be transformed into the polytope representations underlying the algorithms in this thesis [91]. Another assumption made is that the environments are free of moving obstacles. Once again, whether or not this is problematic is dependent on the use case. One strategy of dealing with moving obstacles is to allow the planning algorithms to re-plan. A second strategy is to use the region planner search tree developed in Chapter 4 to "hot-start" the search if a previous trajectory was invalidated by a moving obstacle.

Chapter 3

Optimal Hybrid Search Via Bounding Expected Cost

This chapter focuses on an approach to optimally solving the planning problem introduced in the proceeding chapter using hybrid search. The chapter concentrates on the risk-bound shortest path problem where an agent must find the shortest path from an initial state to a goal state while avoiding obstacles with some specified probability. Because of its widespread applicability, the deterministic shortest path problem is one of the most studied problems in autonomous systems and robotics; its stochastic counterpart has seen interest recently.

As described in Chapter 2, the deterministic problem is hard because the workspace is non-convex in the presence of obstacles. The stochastic problem must also consider the expected path cost in order to evaluate the goodness of various paths. In general, this quantity is difficult to compute and numerical methods must be used. Furthermore, the algorithm must approximate collision risk. While this chapter presents a basic method for approximating collision risk, it is not the chapter's focus and more sophisticated methods are developed in later chapters.

The problem driving this chapter is that there is often a trade-off between the precision with which the stochastic model is evaluated and the computational effort required to evaluate the model. For example, computing the expected path cost typically requires integrating a probability distribution:

$$\mathbb{E}_Q\left[f_0\left(\mathbf{x},\mathbf{u}\right)\right]=\int_Q f_0\left(\mathbf{x},\mathbf{u}\right)q\left(\mathbf{x}\right)d\mathbf{x} \tag{3.1}$$

$$\int_Q f_0\left(\mathbf{x},\mathbf{u}\right)q\left(\mathbf{x}\right)d\mathbf{x}\approx\sum_n^N w_n f_0\left(\mathbf{x},\mathbf{u}\right) \tag{3.2}$$

where f_0 is a cost function, $\mathbf{u}$ a set of control inputs, and Q is some distribution of the agent's state $\mathbf{x}$. The integral in (3.1) is typically difficult to compute analytically and a numerical integration or quadrature scheme is used, e.g. Eq. (3.2) where w_n are weights. A wide variety of schemes are possible, most have the property that their error is reduced as $N \to \infty$. This creates a challenge for path planning algorithms. Increasing the resolution of the numerical model provides more accuracy but it also means a slower search. It is difficult to set N, i.e. the precision with which we evaluate the stochastic cost, in order to ensure the algorithm outputs the optimal path and that the algorithm runs quickly.

The insight behind the approach presented in this chapter is that bounding can be used to help determine N at various stages of the algorithm. The general strategy is as follows. The search for an optimal path begins by evaluating path costs with low precision (i.e. small N values). If the precision is too low and it is ambiguous whether one path is better than another, the precision is increased. The search proceeds in this manner until an optimal path is produced. In summary, there are three main contributions in this chapter:

- A product formulation of the chance constraint allows the problem to be modeled as a convex program; the formulation is less conservative and simpler than previous convex methods.

- A method for bounding the expected path cost; the computation can be easily scaled when more precision is required.

- A state space search that computes good paths quickly and effectively scales computational effort to prune.

3.1 Problem Statement

This chapter investigates a subset of the motion planning problem discussed in Chapter 2:

$$\begin{aligned}
\text{minimize} \quad & \mathbb{E}\left[f_0\left(\mathbf{x}_{0:K}, \mathbf{u}_{0:K-1}\right)\right] && (3.3)\\
\text{subject to} \quad & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{w}\ \forall\ k \in K && (3.4)\\
& P\left(\bigwedge_{k \in K} \mathbf{x}_k \in \mathcal{W}_{free}\right) \geq 1 - \epsilon && (3.5)\\
& \mathbf{x}_0 \in \mathcal{I} && (3.6)\\
& \mathbf{x}_T \in \mathcal{G} && (3.7)
\end{aligned}$$

The main simplification is the linear dynamics with additive noise, Eq. (3.4), where $\mathbf{w} \sim \mathcal{N}(\mathbf{u}, \Sigma)$. The chapter focuses on the objective, (3.3), and how it can be efficiently computed while maintaining the dynamics constraints (3.4) and chance constraint (3.5). The chapter focuses on the case when f_0 is convex. This is important in the case of solving shortest path problems when f_0 may be the Euclidean distance between trajectory knot points.

3.2 Hybrid Search Framework

The search presented in this chapter relies on the hybrid search framework presented in Chapter 2. The components of the region and trajectory planners are described in the following subsections.

3.2.1 The Region Planner

In order to define the region planner, it is necessary to define the form of the regions $\mathcal{R}$. For this chapter, I assume the existence of an algorithm that can produce a complete partitioning of the workspace $\mathcal{W}$ into convex sub-regions $\mathcal{W}_n$. A complete partitioning means that each sub-region of the workspace $\mathcal{W}' \in \mathcal{W}$ can be mapped to exactly one

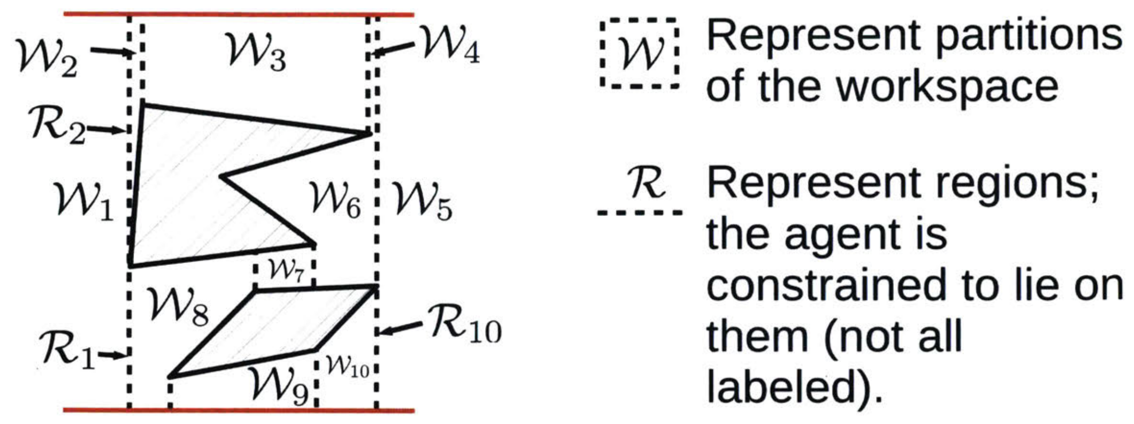


Figure 3-1: Trapezoidal decomposition is a simple technique to form the adjacency graph used by the region planner. Vertical lines are drawn from each polygon vertex until they collide with another obstacle or the environment's boundary. Neighboring regions form nodes in the graph.

$\mathcal{W}_n$. In general, such a complete partitioning of the workspace into convex sub-regions can be very challenging to compute [30]. The hardness depends on the problem's domain and whether there is a natural partitioning of $\mathcal{W}$. This chapter discusses two use cases: a car navigating city streets and an underwater vehicle avoiding undersea obstacle. A car navigating city streets has a natural partitioning: streets are modeled as rectangles and intersections as squares. These can be trivially partitioned into convex $\mathcal{W}_n$.

On the other hand, the underwater domain does not naturally lead to a complete partitioning. Instead, simplifications are needed as well as an algorithm to partition the space. This chapter assumes that the workspace is 2D and the obstacles can be modeled as trapezoids. Under these assumptions, a trapezoidal decomposition algorithm is utilized as described in [8]. The trapezoidal decomposition proceeds by drawing lines of constant x from each of the vertices of the polygon obstacles. The lines are drawn until they intersect the edge of another trapezoid or the environment border. The process is termed a trapezoidal decomposition because the convex sub-regions that are formed are trapezoids. While the trapezoidal decomposition is fast, even for a simple environment, a relatively large number of trapezoids are generated. The geometry is illustrated in 3-1.

At this point, it is possible to define the regions $\mathcal{R}$, which are necessary to carry out the hybrid search. Regions are generated such that they lie at the interface between $\mathcal{W}_n$ as shown in Fig. 3-1. Geometrically, the $\mathcal{R}$ are line segments, i.e. $\mathcal{R} \in \mathbb{R}^1$ of constant x. For the search described later in the chapter, it is necessary to define a *successor* relationship for each $\mathcal{R}$ in order to expand nodes in the search tree. Given a parent $\mathcal{R}_n$, the children are simply the regions $\mathcal{R}_m$ which share a $\mathcal{W}_n$ with $\mathcal{R}_n$. In the case where the x component of the goal state is greater than the x component of the initial state, the successors are always chosen with *increasing* x value. The opposite is true if the goal state is at a lower x value than the initial state. This second condition on the second relation is due to the output of the trapezoidal decomposition algorithm; it guarantees that progress is always being made towards the goal.

The region planner passes a sequence of $\mathcal{R}$ to the trajectory planner.

3.2.2 The Trajectory Planner

Given an ordered set of $\mathcal{R}$, the trajectory planner optimizes a trajectory $\mathcal{T}$ where the trajectory is constrained to pass through the $\mathcal{R}$. For the test cases discussed in this chapter, the form of the constraints is problem-dependent. In the undersea obstacle case, it was mentioned above that the trapezoidal decomposition algorithm produces a set of trapezoidal regions bounded by lines of constant x and that each $\mathcal{R}_n$ are line segments of constant x. Therefore, the trajectory is constrained at various knot points to lie on the $\mathcal{R}_n$, i.e. be at a certain x value. This construction is depicted in Fig. 3-2.

In the case of the road map test case, the trajectory is constrained to lie at the intersection of the streets. Because the intersections are modeled as squares, the regions constraining the trajectories are convex and the constraints required in the trajectory optimization are linear:

$$a_i x + b_i y + c_i \geq 0 \quad i \in \{1, 2, 3, 4\} \tag{3.8}$$

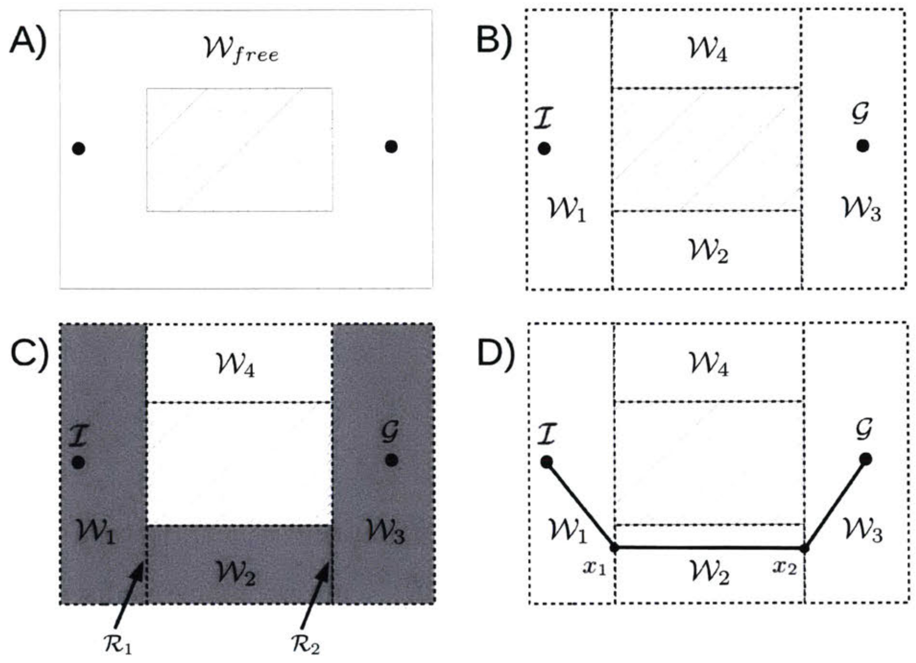


Figure 3-2: The solution to the optimal control problem proceeds in stages. The free region of the workspace, $\mathcal{W}_{free}$, is parsed into convex subregions (B) and state space search determines the best set of regions (C). An inner path planner determines the best path passing through a set of regions (D).

where a_i, b_i, and c_i are constants determined by the orientation of the square. There is a constraint for each of the square's four sides.

3.3 Stochastic Objective

The risk-aware path planning problem considers expected rather than deterministic cost. Computing this is often intractable because of a multidimensional integration:

$$\mathbb{E}\left[f_0\left(\mathbf{x},\mathbf{u}\right)\right] = \int f_0\left(\mathbf{x},\mathbf{u}\right) q\left(\mathbf{x}\right) d\mathbf{x} \tag{3.9}$$

where the expectation is taken with respect to some distribution q. For this work, additive cost is considered. The total expected path cost is given by the sum of the expected costs of the transitions:

$$\mathbb{E}\left[f_0\right] = \sum_t \mathbb{E}\left[f_{0,t}\right] = \sum_t \mathbb{E}\left[f_0\left(\mathbf{x}_t, \mathbf{x}_{t+1}\right)\right] \tag{3.10}$$

While the cost function in Eq. (3.10) may be an explicit function of $\mathbf{u}$, we assume for the remainder of the work that it is only explicit in the state $\mathbf{x}$, e.g. the Euclidean distance between waypoints.

For specific forms of the cost function f_0, (3.10) has a tractable analytic representation such as when $f_0\left(\mathbf{x}\right) = \mathbf{x}$, $\mathbb{E}\left[f_0\left(\mathbf{x},\mathbf{u}\right)\right] = \mathbb{E}\left[\mathbf{x}\right] = \bar{\mathbf{x}}$. The well-known Linear Quadratic Gaussian takes advantages of a quadratic form: $\mathbb{E}\left[f_0\left(\mathbf{x},\mathbf{u}\right)\right] = \mathbb{E}\left[\mathbf{x}Q\mathbf{x}\right] = \bar{\mathbf{x}}Q\bar{\mathbf{x}} + tr\left(Q\Sigma_{\mathbf{x}}\right)$ [86]. While relying on a cost function with an analytic representation makes the problem much more tractable, there are two main drawbacks: it precludes a large number of useful cost functions such as the Euclidean distance and it prevents integration over complicated geometries such as found in the obstacle avoidance case.

Numerical approximation is useful when analytic solutions do not exist. Sampling approaches are one possible numerical approximation. These approaches draw N

samples from distribution q and evaluate the cost of each sample.

$$\hat{f}_0 = \frac{1}{N} \sum_{i}^{N} \sum_{t}^{T} f_0 \left(\mathbf{x}_{s,t}, \mathbf{x}_{s,t+1} \right) \tag{3.11}$$

Eq. 3.11 has the desirable property that $\hat{f}_0 \to \mathbb{E}[f_0]$ as $N \to \infty$. While simple, sampling approaches pose the difficulty of selecting N; too few samples will yield an incorrect result and too many will make computation difficult.

To solve the search problem posed by the region planner, there are two desirable properties for approximating $\mathbb{E}[f_0]$. First, the approach must allow coarse approximations to be quickly computed and allow increasing precision as needed by the search. Second, the approach must allow for path costs to be easily compared so that the search may prune nodes. Toward these objectives, we propose a numerical bounding approach. Lower and upper bounds on the expected path cost are developed in the next sections. The search presented in Section 3.4 uses these bounds to solve the optimal control problem.

3.3.1 Bounding the Expected Cost

The goal is to construct lower and upper bounds of the expected trajectory cost, Eq. (3.10). For the lower bounds, the well-known Jensen's bound is used. For an upper bound, a form of the Edmundson-Madansky inequality is proposed. The bounds assume convex f_0; the cost function must be convex for the path planning problem to be convex.

Computing a Lower Bound: To compute the lower bound of Eq. (3.10), Jensen's inequality is used. For convex f_0:

$$f_0 \left(\mathbb{E}[\mathbf{x}] \right) \leq \mathbb{E}\left[f_0 (\mathbf{x}) \right] \tag{3.12}$$

Considering the Euclidean cost in two coordinates, Eq. (3.12) becomes:

$$\mathbb{E}\left[f_0 \left(\mathbf{x}_t, \mathbf{x}_{t+1} \right) \right]_{lb} = f_0 \left(\mathbb{E}[\mathbf{x}] \right) = \sqrt{\left(\bar{x}_t - \bar{x}_{t+1} \right)^2 + \left(\bar{y}_t - \bar{y}_{t+1} \right)^2} \tag{3.13}$$

Computing an Upper Bound: The Edmundson-Madansky bound is used for the upper bound. This bound uses the fact that a linear interpolation of a convex function is an upper bound on the function [71]. Second, the linearity of expectation is used to enable quick computation of the interpolation's expected value. For example, in the two dimensional case, a linear interpolation is given by:

$$f(x_1, x_2) = a + bx_1 + cx_2 + dx_1x_2 \tag{3.14}$$

where a, b, c, and d are interpolation constants. If the original function is convex, then the interpolation, Eq. (3.14), is an upper bound. Taking the expected value of the interpolation and using the linearity of expectation yields the two dimensional upper bound.

$$\begin{aligned}\mathbb{E}\left[f(x_1, y)\right] &= a + b\mathbb{E}\left[x_1\right] + c\mathbb{E}\left[x_2\right] + d\mathbb{E}\left[x_1\right]\mathbb{E}\left[x_2\right] &(3.15)\\ &= a + b\bar{x}_1 + c\bar{x}_2 + d\bar{x}_1\bar{x}_2 &(3.16)\end{aligned}$$

where the independence of x_1 and x_2 is used. The interpolation constants can easily be found using a number of interpolation schemes. This work uses Newton's divided difference algorithm. In the case of a bivariate transition, there are four state variables $(x_t, y_t, x_{t+1}, y_{t+1})$. The full linear interpolation is given by the summation:

$$\mathbb{E}\left[f_0(\mathbf{x}_t, \mathbf{x}_{t+1})\right]_{ub} = \sum_{i,j,k,l=0}^{1} c_{i,j,k,l}x_t^i x_{t+1}^j y_t^k y_{t+1}^l \tag{3.17}$$

where $c_{i,j,k,l}$ are interpolation constants. Eq. (3.17) provides an upper bound on the expected cost for the case of bivariate uncertainty.

Improving the Bounds: Given the above expressions that bound the expected path cost, a discretization is performed that allows bound tightness to be increased. This is done by discretizing the support of q into R rectangles. If q has infinite support (if q is Gaussian), a region is selected that contains a large portion of the probability mass. For each discretization, expected cost bounds are computed using Eqs. (3.13)

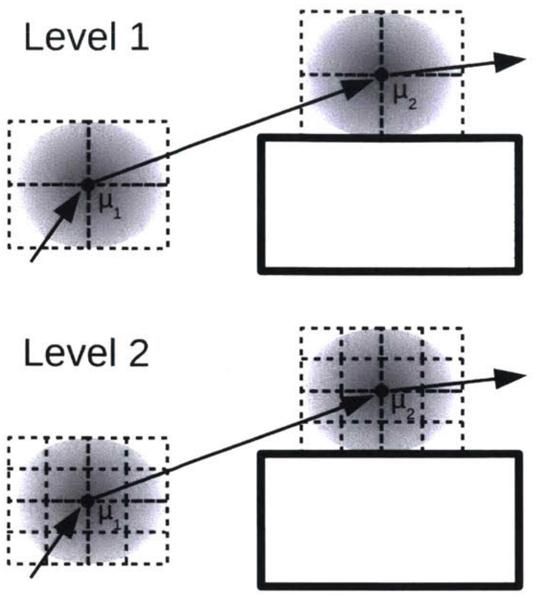


Figure 3-3: Discretizing the support of f to approximate the bounds on the expected transition cost. With an increased number of discretizations, the bounds tighten for a more precise estimate of $\mathbb{E}[f_0]$.

and (3.17). Instead of using mean values for the entire distribution in these equations, means must be computed for each discretization. This is straightforward for Gaussian uncertainty where the truncated Gaussian models $\mathbb{E}[x|a < x < b]$. Specifically, for a region bounded by a and b and conditioned on $a \leq x \leq b$, the expected value of x has an analytic expression [22]. Each bound is computed as the sum over all discretizations, $\mathbb{E}[f_0]_{lb} = \sum_r \mathbb{E}[f_0]_{lb,r} < \mathbb{E}[f_0] < \sum_r \mathbb{E}[f_0]_{ub,r} = \mathbb{E}[f_0]_{ub}$.

For continuous convex f_0, as the number of rectangles R increases, the bounds on expected cost become tighter. The discretization process is illustrated in Fig. 3-3 and the performance of the bounding approach is compared in Fig. 3-4. It performs similarly to the sampling approach with the key advantage that it provides an explicit bound. Importantly, the entire procedure of computing bounds on the expected cost can be parallelized. This provides a substantial improvement because the optimizer must approximate $\mathbb{E}[f_0]$ many times.

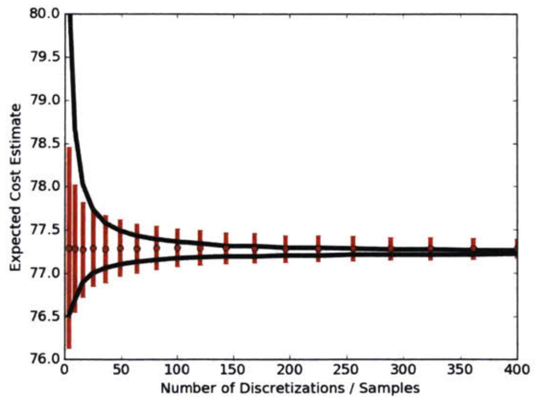


Figure 3-4: Performance of the expected cost approximations for a 25 waypoint path where f_0 is the Euclidean distance. Black lines indicate simulations of Eqs. (3.13) and (3.17) as a function of R, the number of discretizations. Red error bars indicate confidence intervals for the sampling approach, Eq. (3.11), that capture 99% of simulated values.

3.4 Depth-First Approximate Branch & Bound

To solve the optimal control problem, state space search is used to determine the optimal set of regions that allow the agent to travel from the initial state to the goal state. First, terminology is reviewed. A path $\mathcal{P}$ is an ordered set of neighboring regions, i.e. $\mathcal{P} = [\mathcal{R}_1, ..., \mathcal{R}_T]$. A *complete* path connects the initial state to the goal state, $\mathcal{I} \in \mathcal{R}_1, \mathcal{G} \in \mathcal{R}_T$, and a *partial* path does not reach the goal. Path $\mathcal{P}_1$ *dominates* $\mathcal{P}_2$ if $\mathbb{E}[f_0]_{ub,1} < \mathbb{E}[f_0]_{lb,2}$. If a path does not dominate another, they are said to be *ambiguous*.

Input to the search is a set of connected convex regions, each assigned one admissible heuristic value that underestimates the cost to go from the region to the goal. The basic search strategy is similar to A*: candidate paths are popped from a priority queue. Candidates are expanded by adding the neighboring regions and evaluating the new path's expected cost.

The search is made more efficient by exploiting the insights that paths should be expanded to termination and only terminal paths tested for domination by other paths. The heuristic can be a considerable underestimate of path cost - if a deterministic cost estimate is used, the underestimate grows with increased variance of $\mathbf{w}$.

Using A* with standard priority $f = g + h$ will bias shorter paths because they are dominated by h; A* will tend to expand poor quality but short paths and become inefficient. Instead, our approach explores candidate paths in a depth-first manner until the path is either terminal or can be pruned by a previously expanded terminal path. Once a candidate path is terminal or pruned, a different path is popped using the A* priority of $g + h$. Due to the strategy of exploring paths depth-first and our ability to approximate path cost, we name the search Depth-First Approximate Branch & Bound (DF-ABB).

The complete algorithm is given in Algs. 1 and 2. The search proceeds in three phases. The outermost loop maintains a priority queue, *GlobalQueue*, of candidate paths. The global queue uses the A* priority: $f = g + h$ where $g = \mathbb{E}\left[f_0\right]_{lb}$ and h is a deterministic cost estimate. A candidate path *cand* is popped and its children expanded. The child's feasibility is checked with respect to the chance constraint. This check is straightforward by considering the most conservative path for a set of regions, i.e. a path that travels furthest from the obstacles. Line 16 checks whether the child is cached or can be pruned using the subroutine $PruneTest$. If this check passes, it is added to *LocalQueue* to ensure the current path is explored in depth-first order. A dominated child is returned to *GlobalQueue*. In line 6, terminal paths are compared against an incumbent to determine if they replace the incumbent. The process continues until the best candidate on *GlobalQueue* is dominated.

3.4.1 Path Extensions & Pruning

Because the search continues until all paths are dominated, it is important that DF-ABB prune paths often. As noted above, only terminal paths should be considered for pruning to avoid favoring shorter paths. Instead of searching all paths to termination, which would be intractable, DF-ABB extends partial paths by caching previously-explored terminal paths. The extension allows the current path to be compared against prior paths and one of the paths pruned.

First, terminal paths are cached. While it is possible to cache paths based on their set of regions, we found it more effective to cache based on the active constraint set.

Defining an active constraint is problem specific; for obstacle avoidance we used the closest obstacle to a waypoint. The process proceeds as follows. Let $A_{T'}$ be the terminal active constraint of partial path $\mathcal{P}_p$ of length T'. Assume a previously-explored terminal path $\mathcal{P}_{term}$ of length T has passed through $A_{T'}$. $\mathcal{P}_p$ is extended to yield a pseudo-terminal path, $\mathcal{P}_p' = \mathcal{P}_p + P_{term,T':T}$. The path planning optimization is resolved for $\mathcal{P}_p'$ and the cost compared against $\mathcal{P}_{term}$. There are three possibilities: $\mathcal{P}_{term}$ dominates $\mathcal{P}_p'$, $\mathcal{P}_p'$ dominates $\mathcal{P}_{term}$, or the result is ambiguous. If $\mathcal{P}_{term}$ dominates $\mathcal{P}_p'$, the extended path $\mathcal{P}_p'$ is returned to the global queue and a new path popped. If $\mathcal{P}_p'$ dominates $\mathcal{P}_{term}$, then it replaces $\mathcal{P}_{term}$ in the cache and we continue its expansion in depth-first order. If the two paths are ambiguous, the level of discretization is increased to tighten the bounds on $\mathbb{E}\left[f_0\right]$ and resolve the ambiguity.

The process is described in Alg. 2. Line 5 resolves the ambiguity between paths by increasing the precision with which $\mathbb{E}\left[f_0\right]$ is computed. In implementation a maximum-precision threshold is used for *ResolveAmbiguity*, i.e. two paths may be truly equal in cost or very close to equal. In this case the algorithm is ambivalent to which path is cached.

A subtle detail is that even if dominated, $\mathcal{P}_p$ is not pruned outright but returned to the global queue. The reason is that the best path from T' to T may change and yield a better path with which to extend $\mathcal{P}_p$.

3.5 Numerical Benchmarks

This chapter performs a number of benchmarks against various chance constraint and search models. A benchmark is performed of the DF-ABB algorithm against a sampling-based approach. The chance constraint models are introduced briefly as methods to ensure that the trajectory is risk-bound; they will be explored more fully in Chapter 6.

Algorithm 1: DF-ABB

```
Input: A non-convex chance-constrained optimal control problem
Output: An open loop controller, $\mathbf{u}$
1  $Regions \leftarrow GenRegions(Obstacles)$
2  $DetermineHeuristicCosts(Regions)$
3  $GlobalQueue \leftarrow GetTreeRoot(Regions)$
4  while $\neg cand.Dominated()$ do
5      if $cand.IsComplete()$ then
6          $TestAgainstIncumbent(cand)$
7      if $LocalQueue.IsEmpty()$ then
8          $cand \leftarrow GlobalQueue.Pop()$
9      else
10         $cand \leftarrow LocalQueue.Pop()$
11         $GlobalQueue.Add(LocalQueue)$
12         $LocalQueue.Empty()$
13     for $ch \in cand.children$ do
14         if $ch.Feasible$ then
15             $ch.UpdateCost()$
16             if $\neg ch.Cached() \lor \neg PruneTest(ch)$ then
17                 $LocalQueue.Add(ch)$
18             else
19                 $GlobalQueue.Add(ch)$
20     end
21 end
```

3.5.1 Chance Constraint Assuming Independence

A described in Chapter 2, the chance constraint models the risk of mission failure and is difficult to compute in the general case as represented by Eq. (3.5):

$$P\left(\bigwedge_{k\in K} \mathbf{x}_k \in \mathcal{W}_{free}\right) \geq 1-\epsilon \tag{3.18}$$

Various techniques have been developed in order to simplify the constraint. Especially difficult to model are the dependencies between various time steps. If the distribution of the agent's state at various time steps is taken as independent, the chance constraint can be simplified as follows:

$$P\left(\bigwedge_{k\in K} \mathbf{x}_k \in \mathcal{W}_{free}\right) \approx \prod_{k\in K} P\left(\mathbf{x}_k \in \mathcal{W}_{free}\right) \tag{3.19}$$

Algorithm 2: Subroutine *PruneTest*

```
Input: An extended partial path, $PP$, a cached path $CP$ to compare against
Output: Decision to add $PP$ to $LocalQueue$ or $GlobalQueue$
1 $pop\_global\_queue = True$
2 if $PP.f\_cost\_ub \leq CP.f\_cost\_lb$ then
3     $pop\_global\_queue = False$
4 else if $PathsAmbiguous(PP, CP)$ then
5     $ResolveAmbiguity(PP, CP)$
6     if $PP.f\_cost\_ub \leq CP.f\_cost\_lb$ then
7         $pop\_global\_queue = False$
8 if $\neg pop\_global\_queue$ then
9     $UpdateCachedPaths(PP)$
```

Eq. (3.19) is still very difficult to evaluate because it requires the state distribution to be integrated with respect to $\mathcal{W}_{free}$, a potentially complex geometric region. If we assume the risk is only evaluated with respect to a region constraint $\mathcal{R}$, the computation becomes much simpler because $\mathcal{R}$ is convex:

$$\prod_{k \in K} P\left(\mathbf{x}_k \in \mathcal{W}_{free}\right) \approx \prod_{k \in K} P\left(\mathbf{x}_k \in \mathcal{W}_{free}\right) \geq 1 - \epsilon \tag{3.20}$$

If it is assumed that $\mathcal{R}$ take the form of line segments as in Fig. 3-1 and that uncertainty is Gaussian, evaluating Eq. (3.20) simply requires the evaluation of a product of Gaussian cumulative distribution functions:

$$\prod_{k \in K} P\left(\mathbf{x}_k \in \mathcal{W}_{free}\right) \approx \prod_{k \in K} \Phi\left(l_{ub}\right) - \Phi\left(l_{lb}\right) \tag{3.21}$$

This is the chance constraint modeled by the chance constraint labeled *nonlinear* in this chapter's results. Its properties are explored more fully in Chapter 6.

3.5.2 Chance Constraint Using Boole's Inequality

One innovative method to deal with chance constraints is decomposing the constraint using Boole's inequality [80]. Boole's inequality can be used to constrain the probability of path failure by summing the probability of failure at each time step. The chance constraint $P\left(\bigwedge_k^K \mathbf{x} \in \mathcal{R}\right) \geq 1-\epsilon$ is implied by the conjunction:

$$\left(\bigwedge_k^K P\left(\mathbf{x}_k \in \mathcal{R}_k\right) \geq 1-\delta_k\right) \wedge \left(\sum_k^K \delta_k \leq \epsilon\right) \tag{3.22}$$

where δ_k models the risk allocated to each individual constraint. The sum of the individual risks must be less than or equal to the total mission risk ϵ. Constraint (3.22) is tractable because it allows the constraint to be written in terms of the inverse distribution function, F^{-1}:

$$\mathcal{R}_k \geq F^{-1}\left(1-\delta_k\right) \implies P\left(\mathcal{R}_k\right) \geq 1-\delta_k \tag{3.23}$$

Constraint (3.23) is convex for Gaussian uncertainty with $\epsilon \leq 0.5$. One inefficiency with Boole's inequality is that it introduces conservativeness due to the assumption that events are mutually exclusive.

3.5.3 Sampling-Based Approaches

To solve the optimal control problem, sampling methods have been used for two approximations: the expected cost and the chance constraint. A method of chance-constrained optimal control is provided in [14] where expected cost and probability of collision are estimated by drawing samples for $\mathbf{w}$.

The chance constraint can be approximated using a finite set of sampled paths that simulate stochasticity. By determining the number of paths that violate an obstacle constraint at least once, the probability of failure may be approximated:

$$1-P\left(S_K\right) \approx \frac{1}{N}\sum_i^N \mathbb{1}_{\mathcal{P}_{i,fail}} \leq \epsilon \tag{3.24}$$

$$\bigvee_k \mathbf{x}_k \notin \mathcal{R} \implies \mathcal{P}_{i,fail} \tag{3.25}$$

Eq. (3.24) approximates the probability of failure $\left(1-P\left(S_K\right)\right)$ by computing the expected value of an indicator variable which is 1 if a sampled path fails. Eq. (3.25) indicates that if any of the individual constraints at time step k are broken, then the entire chance constraint is broken. Because an integer decision variable is required to indicate whether a path has failed or not, the method described in [14] is solved as a mixed integer linear program.

Using Eq. (3.11), the expected path cost can also be approximated through sampling techniques. This approach is discussed in [14]; other work has also explored sampling techniques to evaluate expected path utility [25], [76]. Although these approaches do not discuss parallelizing Eq. (3.11), we do in order to provide a fairer comparison with our bounding approach. Specifically, we construct a benchmark by solving the optimal control problem where the optimizer computes Eq. (3.11) in parallel on a GPU. A drawback of sampling schemes is the lack of explicit bounds on the estimated statistic. Instead of bounding path cost as in DF-ABB, the benchmark uses a baseline A* state space search.

3.5.4 Test Cases

An obstacle avoidance test case was constructed that modeled a Slocum glider navigating through a 2D obstacle field [115]. The Slocum autonomous underwater vehicle (AUV) admits simple linearized dynamics:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} u_{x,t} \\ u_{y,t} \end{bmatrix} + \begin{bmatrix} z_{x,t} \\ z_{y,t} \end{bmatrix} + \mathbf{w}_t \tag{3.26}$$

where $z_{x,t}$, $z_{y,t}$ model deterministic ocean currents and were assumed to be constants for this work. Randomized obstacle fields were generated to model the ocean floor. Regions between obstacles were generated using a trapezoidal map algorithm found in [8]. Obstacles were randomly generated rectangles.

A second test case was performed modeling a double integrator (e.g. a quadrotor) navigating city streets. Road maps were generated from OpenStreetMap [26]. An example of this test case is shown in Fig. 3-6. Regions were individual streets.

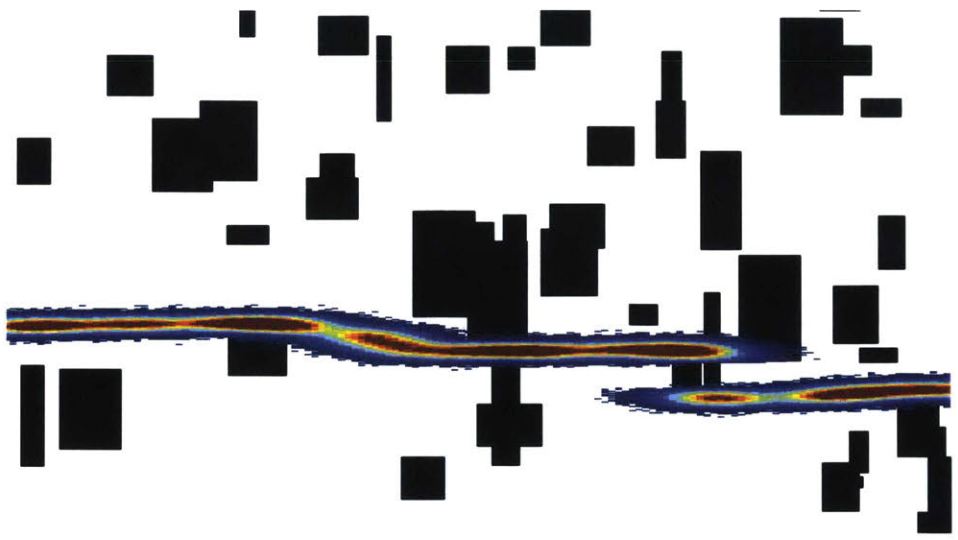


Figure 3-5: Example simulations where the AUV must traverse a 50 obstacle field. The color map denotes the frequency of AUV location over 10,000 simulations.

Waypoints were constrained to lie in intersections and the chance constraint modeled the probability of colliding with a roadside. Both test cases assumed C to be the Euclidean distance. Uncertainty was modeled as a bivariate Gaussian with $\boldsymbol{\mu} = 0$ and various σ_x, σ_y.

Experiments were performed using a GeForce 640 GPU (384 Cuda cores) on a machine with an Intel i7-3770 CPU. A newer GeForce 1080 Ti GPU (3584 Cuda cores) was also used in select tests. Testing the sampling-based chance constraint, Eqs. (3.24), (3.25), required a MILP solver to solve the optimal control problem [41]; all other test cases used a convex solver for their optimizer.

3.6 Results

Tables 3.1 & 3.2 benchmark our chance constraint model and Tables 3.3 & 3.4 compare DF-ABB against the sampling approach. An example simulation of the AUV test case is shown in Fig. 3-5 where the AUV navigates an obstacle field.

As shown in Table 3.1, the proposed chance constraint (modeled by Eqs. (6.8), (6.11), labeled *Nonlinear* in the tables) proves less conservative than using Boole's inequality. The MILP approach (Eqs. (3.24), (3.25), tested using 500 samples per

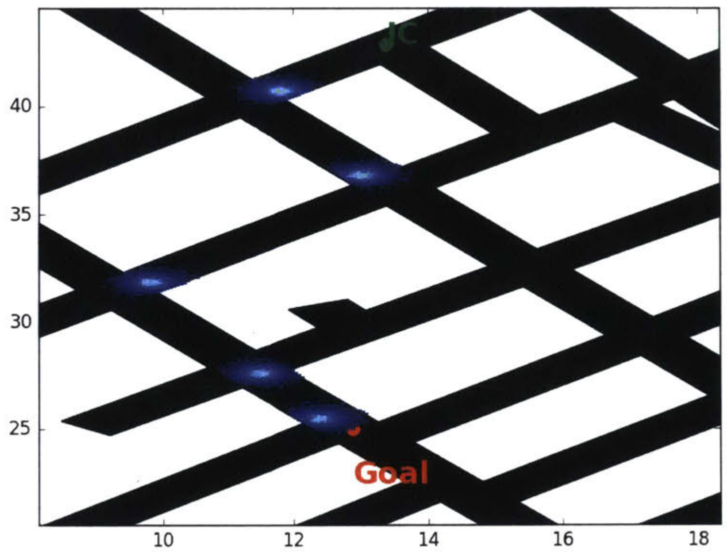


Figure 3-6: Example of algorithm on a road map test case where the road map was generated using OpenStreetMap [26]. Uncertainty ellipses represent vehicle state uncertainty; this was evaluated only at intersections.

waypoint) suffers from chance constraint accuracy and computation time. Because of the integer constraint, the approach is not easily parallelized and leads to mixed integer linear programs with a very large number of decision variables. This results in a slow solve time. Additionally, the MILP had difficulty accurately modeling the chance constraint for longer paths. It is likely increasing the number of samples would improve the results with respect to the chance constraint but this would result in even larger computation time.

Table 3.2 also lists results using a newer GeForce 1080 Ti GPU with approximately ten times the number of cores as the older GPU. While this significantly decreases run times, a considerable amount of time is still spent on search and re-solving the lower level path planning problem. This indicates that both efficient parallelization and search are necessary to quickly solve the non-convex hybrid search.

Finally, the DF-ABB algorithm is compared against the A* parallel sampling approach. Results are presented in Tables 3.3 and 3.4. The sampling approach uses 250 samples per variable per waypoint; empirically, this appeared a good trade-

Test Type	10 Obstacles ($\epsilon = 20\%$)	50 Obstacles ($\epsilon = 20\%$)	75 Obstacles ($\epsilon = 20\%$)	50 Intersections ($\epsilon = 20\%$)	200 Intersections ($\epsilon = 20\%$)
Nonlinear	0.1978	0.2017	0.2019	0.1992	0.1992
Boole's Inequality	0.187	0.188	0.1913	0.1846	0.1846
MILP, N=500	0.1945	0.2039	-	0.1576	0.380

Table 3.1: The ability of various chance constraint formulations to meet the chance constraint ϵ. Results less than ϵ are indicative of conservativeness (i.e. the policy could be improved by being riskier). Failure probabilities were computed after simulating policies 10,000 times.

Test Type	50 Obstacles	75 Obstacles	50 Intersections	200 Intersections
Nonlinear (Geforce 640)	58.77	137.58	8.65	30.19
Nonlinear (Geforce 1080 TI)	40.61	41.38	4.05	8.54
Boole's Inequality (GeForce 640)	122.57	379.97	10.87	43.0
MILP, N=500	3406	-	198.68	1087

Table 3.2: Algorithm run times (in seconds) for various chance constraint models. $\epsilon = 20\%$ for all test cases.

off between accuracy and speed. The tests were performed for various degrees of uncertainty; σ serves as a proxy for this. The DF-ABB performs better both in time and in number of nodes expanded. In both cases, increasing σ increases the difficulty of the problem; as σ increases the heuristic becomes less informative.

The times reported are for the total search time and preclude the trapezoidal decomposition into regions. In no case was the discretization a significant source of time (less than 2-3% of the total runtime in most cases). However, the search time was affected by the underlying decomposition; i.e. if it was too fine then too many waypoints would be placed on the map. Because the trapezoidal decomposition places a vertical line at every obstacle vertex, it can generate a large number of trapezoids, which increases the depth of the search tree. If the branch & bound search performs a full trajectory optimization each time a child is added, the search algorithm can become very slow. A more advanced algorithm would take advantage

σ	A* Sampling time (s)	DF-ABB time (s)	A* Sampling nodes expanded	DF-ABB nodes expanded
0.25	48.5	22.0	645	363
0.50	56.01	36.01	2052	901
0.75	106.15	71.39	2205	1701

Table 3.3: Comparing the A* parallel sampling approach with the DF-ABB approach for various amounts of uncertainty (σ) using a 50 obstacle benchmark.

σ	A* Sampling time (s)	DF-ABB time (s)	A* Sampling nodes expanded	DF-ABB nodes expanded
0.05	16.31	7.99	287	41
0.25	33.76	6.65	302	47
0.35	24.25	11.64	307	193

Table 3.4: Comparing the A* parallel sampling approach with the DF-ABB approach for various amounts of uncertainty (σ) using a 50 intersection benchmark.

of the discretization to make more advantageous waypoint selection. This is the focus of the next chapter.

Chapter 4

Fast Obstacle eXploration (FOX)

The previous chapter developed an approach for optimal path planning under uncertainty using hybrid search. While the method shows the promise of hybrid search, there are drawbacks that prevent it from being useful for more complex path planning problems. Specifically, the hybrid search was only demonstrated on simple 2D environments with convex obstacles. This is because it required a complete partitioning of the workspace into convex subregions. The method of partitioning was problem dependent; I presented two approaches, one for partitioning a set of roadways and another producing a trapezoidal decomposition. Unfortunately, a complete partitioning of the workspace is very hard to produce in the general case.

In this chapter, I argue what is needed to make hybrid search useful to a greater variety of real world problems is an approach to formulate the region planner's $\mathcal{R}$ in a way that is generalizable to a wide range of obstacle types and enables the fast computation of feasible solutions. To do this, this chapter changes from focusing on producing globally optimal solutions to producing good, feasible solutions quickly. The idea that many applications require good, feasible solutions quickly rather than provably optimal ones is a common theme in many approaches in robotics and autonomous systems [42].

As mentioned in Chapter 2, even in the deterministic case, adding obstacles makes the motion planning problem much more difficult. This is because obstacles make the workspace non-convex. For the stochastic case, obstacles pose the added difficulty that the chance constraint must be integrated with respect to the potentially complex obstacle surface. Integrating the chance constraint will be the focus of Chapter 6. This chapter focuses on how more complex obstacle geometry, i.e. 3D, non-convex obstacles, may be integrated with the hybrid search approach introduced in Chapter 2.

There are a number of insights that drive this chapter. First, modeling obstacles using polytopes (polygons in 2D, polyhedra in 3D) enables a rich set of environments to be considered and does not place unnecessary restrictions on the obstacles such that they are convex. Second, in all but trivial instances of the planning problem, a set of obstacle surfaces will serve as active constraints for an optimized trajectory. Therefore, the regions should be formulated as constraints with respect to the obstacle surface. Thirdly, real world environments can be incredibly complex with many superfluous obstacles that are irrelevant to finding good motion plans. This means that regions should be easy to generate and only explicitly enumerated as needed by the region planner. Fourth, a key design element of hybrid search is how the region and trajectory planners interact. For risk-bound motion planning, performing a full trajectory optimization is often an expensive operation. The region and trajectory planners should be integrated in a way that reduces the number of calls to the trajectory planner. These insights are more fully developed over the course of this chapter; together, they form an approach that I term FOX, Fast Obstacle eXploration.

4.1 Hybrid Search via Regions

The underlying motion planning problem under uncertainty from Chapter 2 and 3 is restated here (Problem 1):

$$\begin{aligned} \text{minimize} \quad & \mathbb{E}\left[f_0\left(\mathbf{x}_{0:K}, \mathbf{u}_{0:K-1}\right)\right] && (4.1) \\ \text{subject to} \quad & \dot{\mathbf{x}} = f\left(\mathbf{x}, \mathbf{u}\right) + g\left(\mathbf{w}\right) && (4.2) \end{aligned}$$

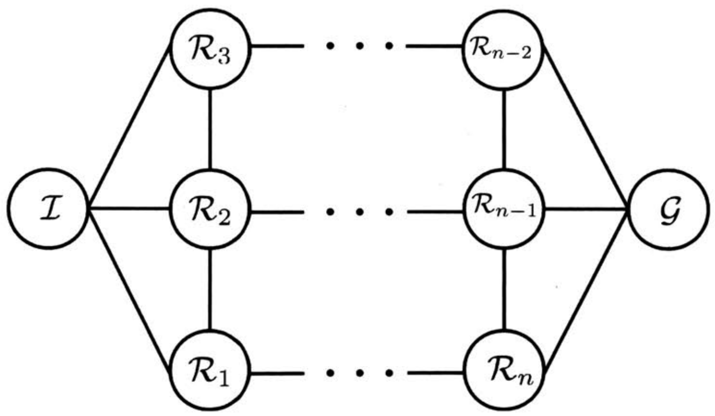


Figure 4-1: Viewing the hybrid search's region planner as a form of graph search. Graph vertices represent regions of the workspace $\mathcal{R}$. Three features define the region planner: the form of the regions, how the regions are connected, and how the graph is searched to form sequences of regions.

$$P\left(\bigwedge_{k\in K} \mathbf{x}_k \in \mathcal{W}_{free}\right) \geq 1-\epsilon \tag{4.3}$$

$$\mathbf{x}_0 \in \mathcal{I} \tag{4.4}$$

$$\mathbf{x}_K \in \mathcal{G} \tag{4.5}$$

As discussed in Chapter 2, the chance constraint, term (4.3), makes Problem 1 very difficult. The general form of this constraint is intractable to compute because it involves integrating a complex state distribution with respect to challenging environment geometry. Even in the case where the state is deterministic, the general case of the collision avoidance constraint is challenging and instances of the problem have been shown to be P-SPACE complete [52], [58].

The general strategy behind hybrid search is to simplify Problem 1 by introducing regions $\mathcal{R}$ that partition the workspace in some way. The region planner generates the regions and joins them together into sequences (i.e. *paths* in the terminology of this thesis) that constrain the trajectory and enable the trajectory planner to output feasible trajectories with respect to the constraints in Problem 1. The region planner's task of constructing the regions and searching for good sequences can be viewed through the lens of constructing and searching an undirected graph, $G =< V, E >$. Vertices V correspond to regions $\mathcal{R}$ and edges E are connections between them; this framework is shown in Fig. 4-1.

There are many forms the regions $\mathcal{R}$ can take. In contrast to the methods presented in Chapter 3, the partitioning of $\mathcal{W}_{free}$ does not need to be complete, i.e. the $\mathcal{R}$ do not have to completely cover $\mathcal{W}_{free}$. Additionally, it is possible that partitions overlap. Regions may also be of various dimension; while many methods assume regions of the same dimensionality as the workspace, this chapter focuses on landmark regions, which are of dimension $d - 1$ for d-dimensional workspaces. Finally, while this thesis uses the term *region* to mean geometric regions of the workspace, similar constructs have been used for the configuration space as well [85]. Approaches from the literature to generating regions are discussed in the next section, Section 4.2.

Given a sequence of regions generated by the region planner, the problem solved by the trajectory planner can be stated as follows for the deterministic case:

$$\begin{aligned} \text{minimize} \quad & f_0\left(\mathbf{x}_{0:K}, \mathbf{u}_{0:K-1}\right) && (4.6)\\ \text{subject to} \quad & \dot{\mathbf{x}} = f\left(\mathbf{x}, \mathbf{u}\right) && (4.7)\\ & \boldsymbol{\omega}_n \in \mathcal{R}_n \;\; \forall \;\; \mathcal{R}_n \in \mathcal{P} && (4.8)\\ & \mathbf{x}_0 \in \mathcal{I} && (4.9)\\ & \mathcal{R}_N \in \mathcal{G} && (4.10) \end{aligned}$$

Variable $\boldsymbol{\omega}$ is some subset of the agent's state vector, $\boldsymbol{\omega} \in \mathbf{x}$, that is constrained to lie within a particular region. For example, $\boldsymbol{\omega}$ may be the agent's position state, (x, y, z). The problem posed by Eqns. (4.6) - (4.10) is termed Problem 2. The formulation is general to hybrid search and is significantly easier for the trajectory planner to solve than Problem 1 given an appropriate form of the regions. For example, if $\mathcal{R}$ represents some convex region of the agent workspace, it is straightforward to transform Eq. (4.8) into a form usable by the trajectory planner. Constraint (4.8) is stated as a deterministic constraint to include hybrid search methods that do not model stochasticity. If stochasticity is included, the constraint may be rewritten:

$$P\left(\bigwedge_n \omega_n \in \mathcal{R}_n\right) \geq 1 - \epsilon \tag{4.11}$$

As will be discussed in Chapter 6, Eq. (4.11) greatly simplifies the computation of collision risk by providing a method to compute the chance constraint's limits of integration.

While Problem 2 is a simplification of Problem 1, the inclusion of regions poses three challenges. First, the geometric form of $\mathcal{R}$ must be specified as well as how the regions are placed in the workspace. Second, it is necessary to devise a method for connecting various $\mathcal{R}$ into paths, keeping in mind that environments are very complex and likely contain a very large number of candidate regions. Third, the relationship between the region planner and trajectory planner must be specified: how are paths formed and what does the trajectory planner return to the region planner. These issues are investigated in the remaining sections of this chapter. A brief review of prior art in methods to generate $\mathcal{R}$ is provided in the next section, Section 4.2. In Section 4.3.2, I introduce the *landmark* region, a type of region constructed for obstacle-avoidance. How the landmark regions are connected together to form paths is detailed in Section 4.3.3. Finally, a method for searching through paths and the interaction between the region and trajectory planners is described in Section 4.4.

4.2 Strategies to Generate $\mathcal{R}$

A number of region forms are possible for the region planner. This section reviews select strategies taken in literature to generate partitions of the workspace. Because motion and path planning are some of the most studied problems in autonomous systems and robotics, this section can only hope to mention a small subset of the prior art for decomposing environments. Many of the approaches mentioned focus neither on hybrid search nor planning under uncertainty. However, these approaches present valuable insights and many could be integrated into hybrid search frameworks.

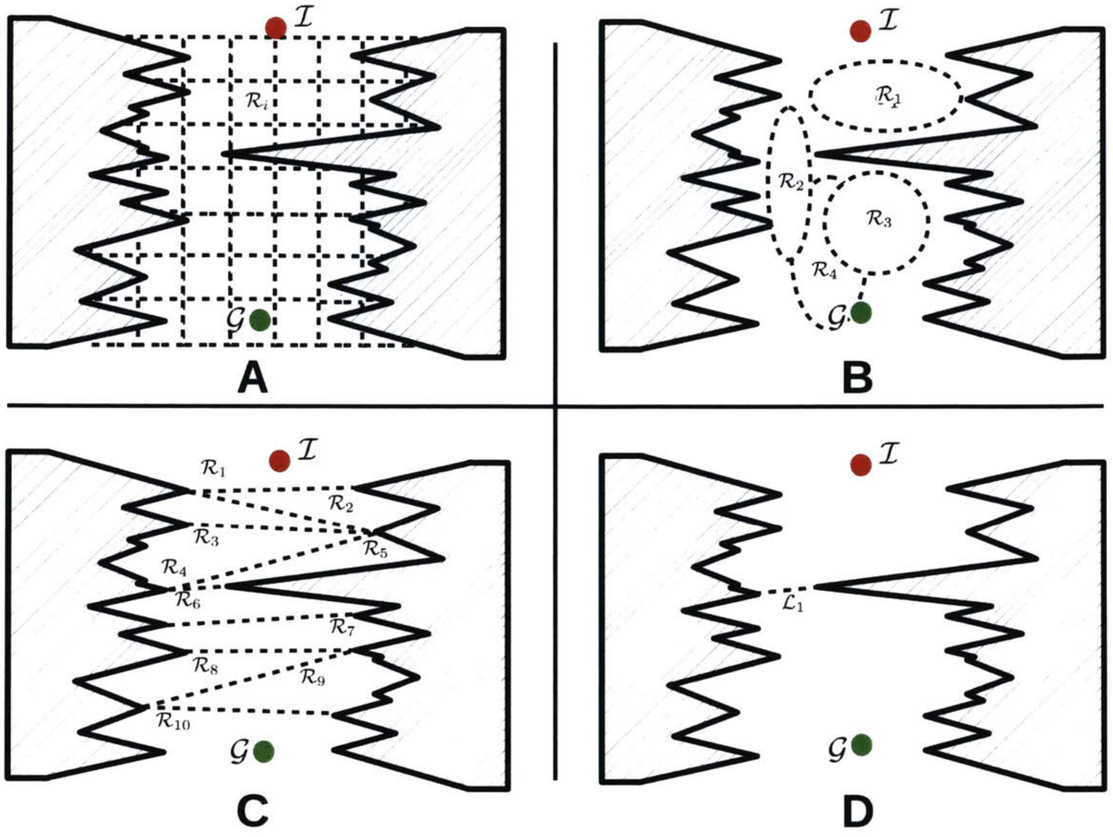


Figure 4-2: An illustration of various methods to partition the workspace. Pane A illustrates a grid-based discretization with a uniform discretization. A challenge of this approach is that the cells may be a poor approximation of the obstacle geometry. Therefore, some approximation scheme is typically necessary where information about the cell, such as whether or not it contains an obstacle, must be averaged over the entire cell. Pane B illustrates methods that attempt to generate convex regions from $\mathcal{W}_{free}$. These approaches have the benefit that paths traveling within a convex region are guaranteed to be collision free, [38],[30]. They have the drawback that a separate method is required to connect the regions and it may be difficult to generate the regions in the presence of non-convex obstacles. Pane C illustrates methods that rely on geometric algorithms such as triangulations or trapezoidal decompositions. Unlike the grid-based methods, these approaches rely on the obstacles' geometry to generate regions. However, they are typically uninformed with respect to the underlying motion planning problem and can generate unnecessary regions. Pane D best illustrates the strategy taken in this chapter. The region is generated based on obstacle geometry. An attempt is made to only generate regions that are relevant to paths navigating from $\mathcal{I}$ to $\mathcal{G}$.

For a review of various methods for robot obstacle-avoidance, the reader is referred to [45].

One of the most popular approaches is to discretize the workspace into partitions, as shown in Fig. 4-2, Pane A [69]. There are many variations of this idea and techniques to generate the discretization. A common method is to use the discretization to produce an occupancy map where each cell relays information about its contents, i.e. whether or not it is obstacle-free [32]. Additionally, many discretization schemes are possible with uniform being the simplest to generate and others attempting to discretize the robot configuration space [120]. The authors in [46] show it is possible to extend the method to 3D workspaces and present the innovative idea of using different map resolutions for modeling the environment and for planning. These methods can be readily applied to kinodynamic motion planning with complex dynamics as shown in [102]. In this work, the focus is on efficiently generating cells to ensure the agent state space is explored and that cells are generated on an as-needed basis. Because the partitioning of the workspace is typically complete, searches that utilize this strategy can often produce globally optimal solutions [69]. In general, grid-based approaches have the downside of choosing an appropriate scale for the discretization. It may be very difficult to determine a priori how many nodes are required for a given problem. Secondly, information about the node's contents, i.e. whether a node contains an obstacle or not, is typically averaged across the entire node. This makes it difficult to approximate the true obstacle surface using this discretization scheme.

Another set of techniques are similar to grid-base occupancy maps but decompose the workspace using geometric features such as obstacle vertices. For example, the Polyanya algorithm performs any-angle path planning on a 2D map that relies on a partitioning of the workspace via a Constrained Delaunay Triangulation. The algorithm is shown to deliver optimal solutions quickly [28]. Another possible set of techniques involve generating regions from every vertex in the workspace. This is similar to the trapezoidal decomposition discussed in Chapter 3. An advantage of these methods is that the algorithms to generate the decomposition are typically efficient. The trapezoidal decomposition and Delaunay triangulation can be computed in $\mathcal{O}(n \log n)$ time [8]. One downside of these strategies is that, similar to grid partitionings, the partitioning is typically done agnostic of the search problem. This means that they can generate unnecessarily complex partitionings. This is an undesirable property for hybrid search because the complexity is exponential in the depth of the search tree. It is especially troublesome for planning under uncertainty because evaluating edge cost (i.e. the cost to travel from one triangulation to another) is expensive. The problem of unnecessarily-generated regions is illustrated in Fig. 4-2, Pane C.

Techniques from topology have also been used to solve the motion planning problem, with a strategy similar to the hybrid search presented in this thesis. Namely, the motion planning problem with obstacles is solved by constraining trajectories to lie inside or outside certain homotopy classes [10]. In general, whether a trajectory belongs to a specific class is dependent on whether there is a smooth deformation that can transform one trajectory into another while remaining in $\mathcal{W}_{free}$. A search is then performed over various classes to determine the best homotopy class for navigating from the initial state to the goal. In comparison with the terminology in this thesis, the search over homotopy classes is similar to the region planner's task; the search over trajectories within a homotopy class can be compared to the task of the trajectory planner. The innovative work has been extended to various use cases such as 3D workspaces [11]. To the author's knowledge, the work has not been extended to cases of planning under uncertainty.

An innovative scheme that relies on semi-definite programming (SDP) to create convex regions (the IRIS algorithm) is proposed in [30]. IRIS is integrated into a hybrid search for multi-agent path planning in [38]. Specifically, ScottyPath is proposed, which combines a region planner that selects sequences of convex regions with a trajectory planner that optimizes a trajectory constrained to lie inside those regions. To generate convex regions from $\mathcal{W}_{free}$, ScottyPath relies on IRIS (Iterative Regional Inflation by Semidefinite programming) [30]. In the case of convex obstacle fields, IRIS is very fast because it can take advantage of efficient SDP solvers. ScottyPath then connects the regions together to form a graph (the connectivity graph, using the terminology of ScottyPath). Graph search is used to produce sequences of convex regions, which are passed to a trajectory planner. In performing the trajectory optimization, the trajectory planner constrains waypoints to lie at the interface of the convex regions. This is a good idea because it means that the trajectory between waypoints will be collision-free. This is in contrast to the disjunctive linear program approach presented in [80], where the trajectory may intersect obstacles between waypoints. There are two main drawbacks to the ScottyPath / IRIS approach. First, it requires computation of the connectivity graph (i.e. region graph using the terminology in this thesis) a priori. For example, the convex regions are computed and connectivity graph generated without reference to the specific instance of the path planning problem. In the case of large obstacle fields, it may be the case that many superfluous regions and graph edges are generated. Second, the algorithm's performance is aided when the region graph forms a complete partition of $\mathcal{W}_{free}$, which is difficult to generate. IRIS generates convex regions by sampling from the free region of the workspace; it is difficult to determine a priori whether the number of samples is sufficient to generate a connectivity graph that allows for a path from the initial state to the goal state. Similar to RRT-type algorithms, coverage of $\mathcal{W}_{free}$ becomes better as the number of samples increases. These problems are alleviated based on the use case; for cases where the environment is static and the agent is not time-constrained in generating the connectivity graph, refined connectivity graphs may be generated and re-used to speed planning.

The approach for constructing regions proposed in this chapter follows three guiding principles. First, regions are constructed at polytope sub-facets (i.e. vertices of polygons in 2D and edges of polyhedra in 3D). This is because these geometric features are likely to be active constraints of an optimized trajectory in the sense that they are likely to cause the trajectory to change its course. It is also possible to use facets as active constraints as they are in a disjunctive linear program [80]. However, a linear trajectory that is only constrained only by polytope facets may still collide with the parent facet. A linear trajectory that is constrained to lie on one side of a parent facet at *each* sub-facet will not collide with the parent facet. Second, the regions should be implicitly defined by the obstacle geometry and simple to generate computationally. That regions are implicitly defined means that they are fully defined by the obstacle geometry alone rather than requiring computation and explicit enumeration. This is a desirable characteristic because it is unknown a prior whether a good path will visit a region. In complex environments, there is likely to be a large number of regions, many of which are irrelevant to the hybrid search. If each region is computationally intensive to compute, the search could slow. Third, it should be straightforward to formulate constraints for the trajectory planner from the regions. This is important for the deterministic motion planning problem (i.e. constraining waypoints to be within a region) and the motion planning problem with uncertainty (i.e. integrating the chance constraint with respect to the region). These features motivate the development of the landmark region, $\mathcal{L}$, as described in the next sections.

4.3 Overview & Notation

An overview of the algorithm and description of the notation is stated here for clarity, repeating some of the descriptions from Chapter 2.

As described above, $\mathcal{R}$ represents a region of the agent workspace $\mathcal{W}$. Sequences of regions form paths $\mathcal{P}$. Regions along paths are indexed by n with the total number of regions being N. As described in Fig. 4-1, the region planner can be viewed as generating an undirected graph of $\mathcal{R}$; this graph is referred to as a *region graph.* The region planner outputs a path to the trajectory planner, which must then produce a trajectory $\mathcal{T}$ that respects the constraints implied by $\mathcal{P}$. Specifically, each region $\mathcal{R}$ implies a set of constraints on a subset ω of the agent's state, $\mathbf{x}$. In the next section, Section 4.3.2, I introduce landmark regions, $\mathcal{L}$. Landmark regions are types of regions that constrain the agent's state to avoid an obstacle surface. The trajectory planner produces an open loop trajectory, denoted by the notation: $\mathcal{T}_{\mathcal{P}}$. As described in Chapter 2, trajectories can take various forms depending on the trajectory optimization framework. In the case of trajectory optimization by direct collocation, where the trajectory

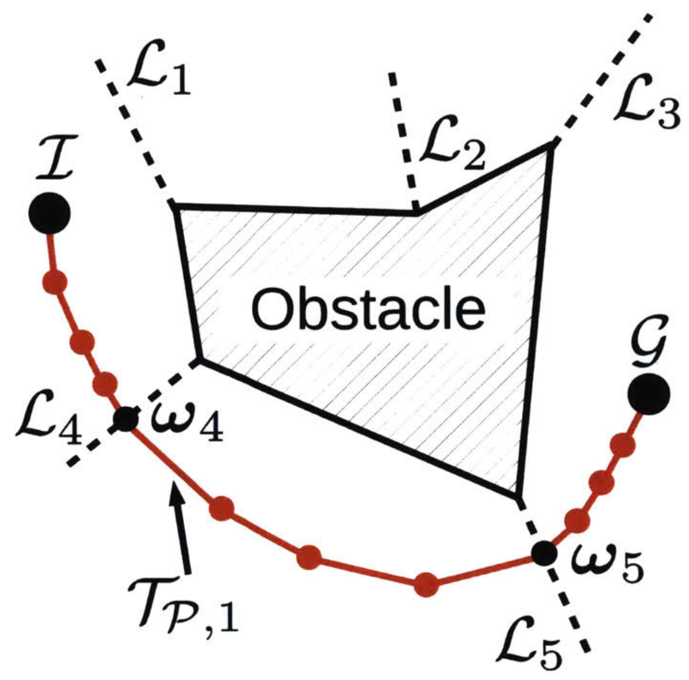


Figure 4-3: Example of the region planner and path planner. Dotted lines represent waypoint constraints; $\boldsymbol{\omega}_4$ and $\boldsymbol{\omega}_5$ are waypoints which must lie on $\mathcal{L}_4$ and $\mathcal{L}_5$. Red dots represent knot points of the trajectory optimization where the trajectory is indicated as $\mathcal{T}_{\mathcal{P},1}$.

is interpolated between various knot points, the trajectory is described by a sequence of states and control inputs indexed by k: $\mathcal{T} = [\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_k, \mathbf{u}_k, \ldots, \mathbf{x}_K, \mathbf{u}_K]$. Each $(\mathbf{x}_k, \mathbf{u}_k)$ is typically referred to as a knot point; the pair can be thought of as the state and control input at a particular point in time [59]. In general, the total number of knots K is considerably greater than the number of waypoints N. If it is the case that $K > N$, only certain knots are constrained to lie on specific waypoints. For example, if there are K_n trajectory knots per waypoint, then the $3K_n^{th}$ knot is constrained by $\mathcal{R}_3$.

The construction is illustrated in Fig. 4-3. The agent must travel from $\mathcal{I}$ to $\mathcal{G}$ while avoiding the obstacle. The region planner generates landmark regions $\mathcal{L}_1 - \mathcal{L}_5$ (not necessarily at the same time). Sequences of regions form paths, $\mathcal{P}$. In this example, the region planner may pass the path $\mathcal{P}_1 = [\mathcal{L}_4, \mathcal{L}_5]$ to the trajectory planner. The trajectory planner will then output $\mathcal{T}_{\mathcal{P},1}$, which respects the geometric constraints implied by $\mathcal{P}_1$ and the dynamical constraints. If the region planner next selects

$\mathcal{P}_2 = [\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3]$, the trajectory optimization is performed, and it is determined that the cost of trajectory $\mathcal{T}_{\mathcal{P},1}$ is less than the cost of $\mathcal{T}_{\mathcal{P},2}$, trajectory $\mathcal{T}_{\mathcal{P},1}$ is output.

4.3.1 Assumptions Regarding the Obstacles

The method to generate constraints is dependent on the geometric representation of the obstacles. The obstacles are assumed to be polytopes, i.e. in three dimensions the obstacles are polyhedra and in two dimensions they are polygons. The polytopes do not need to be convex. Polytopes are used for three reasons. First, polytopes are a linear approximation of a surface and can efficiently represent many real-world objects and surfaces [8]. Second, polytopes enable generation of waypoint constraints that allow for computation of the collision probability, as described in Chapter 6. Thirdly, polytopes greatly simplify the collision-checking process because many efficient algorithms exist to check collisions between polytope facets [33]. Because of this, many approaches in literature also implicitly require polytope-based obstacles [54].

Each polytope is composed of facets, $\mathcal{F}$, which are polygons in 3D and edges in 2D. For each facet, an outward pointing normal vector $\hat{n}$ is assumed known as well as the set of neighboring facets. In the case of 2D obstacles, the neighboring facets are accessed via vertices of the polygon edges. Each vertex has data fields nbr_1 and nbr_2. For 3D obstacles, the halfedge data structure is used. This is a directed edge that lies along each polygon edge of the polyhedral surface. The neighboring polygon facet can be access via a $halfedge.opposite$ relation. This is shown graphically in Section 4.6.

4.3.2 Landmark Regions

Landmark regions are specific types of regions designed to model obstacle-avoidance constraints. Geometrically, my approach uses bounded hyperplanes as landmark regions. In three dimensions, this hyperplane is a plane and in two dimensions, the constraint is a ray. Specifically, in three dimensions, each $\mathcal{L}_n$ takes the form of a

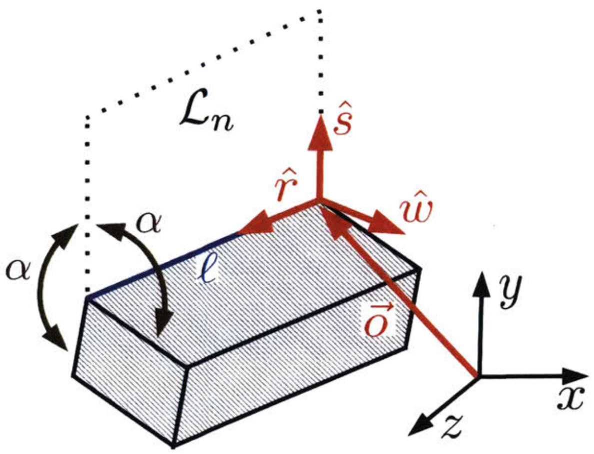


Figure 4-4: The plane and coordinate system that represent a landmark region $\mathcal{L}_n$ for a 3D workspace. The dotted lines do not represent a constraint; they are shown to illustrate the plane.

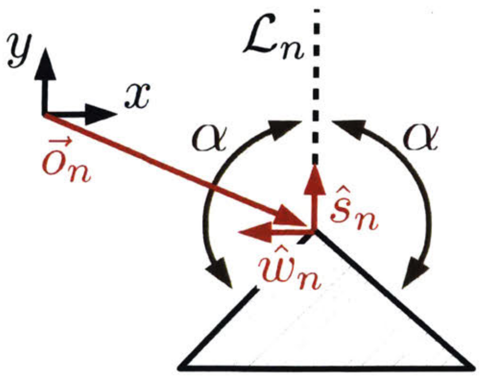


Figure 4-5: The ray that represents a landmark region $\mathcal{L}_n$ for a 2D workspace.

bounded plane. The plane is bounded because it is bordered by a line ℓ_n. The line overlaps an obstacle sub-facet (in 3D the sub-facet is an edge and in 2D the sub-facet is a vertex). Intuitively, the hyperplane radiates outward from an obstacle surface. The geometry is illustrated in Figs. 4-4 and 4-5. The bounded hyperplane extends upwards and in all directions from the blue line. The hyperplane's orientation divides the angle between the two neighboring obstacle facets. If the angle between the facets is 2α (measured on the side of the obstacle surface with outward pointing normal vectors), then the hyperplane is positioned α from each neighboring facet.

An advantage of using a hyperplane is that the transformation from the global position coordinates $\boldsymbol{\omega}_n$ to a coordinate system embedded in the plane is linear:

$$s_n = (\boldsymbol{\omega}_n - \vec{o}_n) \bullet \hat{s}_n \tag{4.12}$$

$$r_n = (\boldsymbol{\omega}_n - \vec{o}_n) \bullet \hat{r}_n \tag{4.13}$$

$$w_n = (\boldsymbol{\omega}_n - \vec{o}_n) \bullet \hat{w}_n \tag{4.14}$$

where $(\hat{s}_n, \hat{r}_n, \hat{w}_n)$ form an orthogonal basis of the embedded coordinate system. Direction $\hat{r}_n$ runs along the line ℓ_n (i.e. it runs along the obstacle's surface), direction $\hat{s}_n$ measures the distance from the obstacle's surface to the waypoint and will be used in Section 6.3.1 to approximate the collision risk. Direction $\hat{w}_n$ is normal to the plane. Vector $\vec{o}_n$ is the offset vector from the origin of the global coordinate system to the origin of the embedded coordinate system. Variables (s_n, r_n, w_n) are the agent's position state transformed into the embedded coordinate system. In order for the waypoint states to lie on the plane, the constraint is necessary:

$$w_n = 0 \;\forall\; n \in N \tag{4.15}$$

In two dimensions, the formulation is analogous. Instead of lying on a plane, the waypoint is constrained to lie on a ray that emanates from a vertex. The 2D embedded basis $(\hat{s}_n, \hat{w}_n)$ is illustrated in Fig. 4-5.

There are a number of insights that motivate the landmark region construction. First, they are completely defined given a polytope surface. This means that they do not require extensive computation or optimization to generate such as can be required by convex decompositions of the workspace. Secondly, they do not place restricting conditions on the obstacle shape or dimensions such as requiring obstacles to be 2D or convex. Third, the number of landmark regions is dictated by obstacle complexity; more complex obstacles will have large numbers of landmarks and simpler environments will have fewer. This contrasts with uniform workspace discretizations typical of occupancy maps where the discretization scale is difficult to select a priori. Finally, the landmark region compiles into simple linear constraints on the agent trajectory.

The challenge is not constructing the landmark regions, but connecting them into paths. This is covered in the next section.

4.3.3 Building the Region Graph

Given the landmark regions described in the preceding section, it is necessary to connect them to form paths through the graph. Specifically, the form of the graph's vertices V has been described in the regions $\mathcal{R}$. This section details how the vertices are connected to form edges E. This work uses the following insights:

1. Environments may be very complex but it is likely that a small number of surfaces will serve as active constraints.

2. The distance to the goal is easy to compute and is available at all areas of the state space; it can direct the search as an admissible heuristic.

3. Obstacle facets and the neighbor relation between facets can be used when the heuristic is wrong.

The first insight relates to the fact that, given a complex environment, the number of edges in the region graph is very large; however, it is likely that only a small number must be enumerated in an optimal path. The second insight can be used to help find candidate good region graph edges with relatively little computation via a heuristic. The final insight relates to the fact that the heuristic can be misleading. For example, the algorithm may need to avoid a concave "bug trap" obstacle; using only the heuristic cost to reach the goal will not work. Using these insights, I discuss two methods to enumerate edges in the region graph. First, a heuristic-based technique is deployed that helps determine which regions may be active on a path to the goal. Second, a loop-detection and facet-iterating scheme is utilized to avoid the case when the heuristic method fails. Other problem-specific edge enumeration schemes are also possible, such as when it is required that an agent travel along a surface.

Heuristic-Based Polytope to the Goal

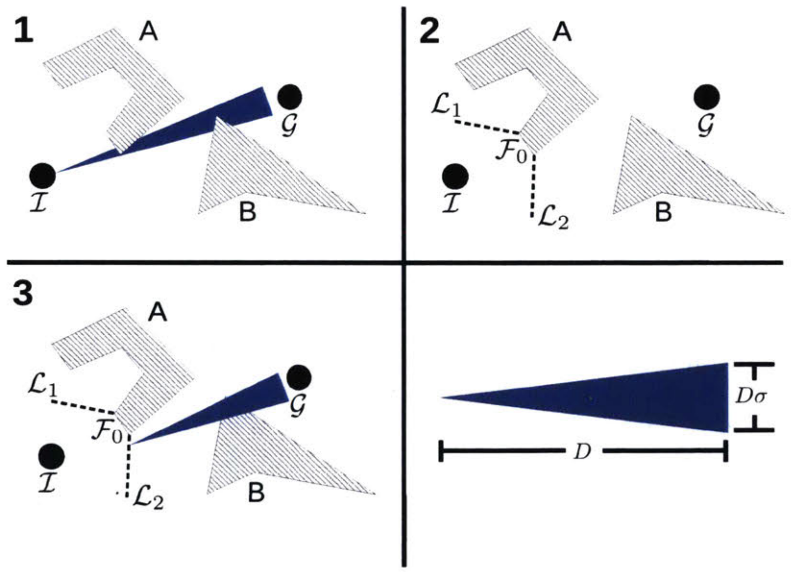


Figure 4-6: An illustration of the heuristic-based method that the region planner uses to determine which landmark regions are likely to be active on good paths to the goal. 1.) A polygon is directed at the goal and a collision checker collects collisions along this polygon. 2.) Regions (i.e. rays in 2D) are formed at the sub-facets of the closest colliding facet. In this case, the closest colliding facet (i.e. edge) is $\mathcal{F}_0$ and $\mathcal{L}_1$ and $\mathcal{L}_2$ at the vertices of $\mathcal{F}_0$ are added to the search tree. 3.) Based on a heuristic cost estimate, the process repeats using $\mathcal{L}_2$ as the parent. 4.) The shape of the polygon approximates state uncertainty, e.g. some multiple of the state uncertainty's variance.

The purpose of the heuristic approach is to connect regions that are likely to lie on good (i.e. low cost) paths to the goal while ignoring obstacles that lie far

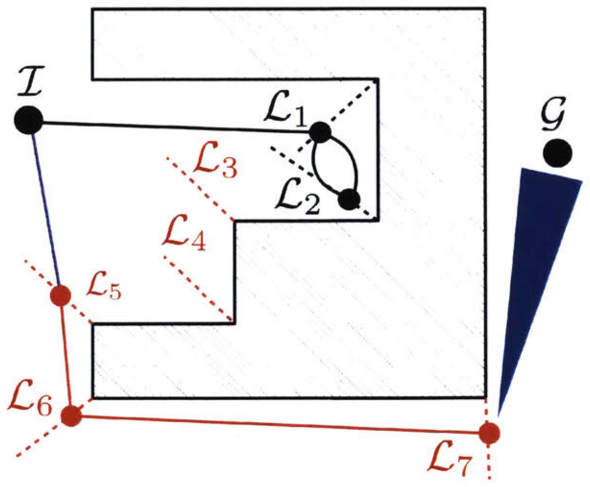


Figure 4-7: $\mathcal{L}_1$ and $\mathcal{L}_2$ will form a loop in the graph using only the heuristic approach; the black line represents the original path. Red constraints $\mathcal{L}_3$ - $\mathcal{L}_7$ are generated so that the path may avoid the sub-facet that generated the loop. The blue line $\mathcal{I}$ - $\mathcal{L}_5$ represents how the graph may be rewired.

off course. The approach consists of shooting a polytope to the goal (abbreviated *PtG*) and detecting collisions along this polytope. The polytope is an approximation of the agent's position state distribution if it were to travel directly to the goal. For example, if the agent's position uncertainty grows proportionally to the distance traveled D and the state standard deviation σ, the width of the PtG's base may be proportional to $D\sigma$. The closest colliding facet $\mathcal{F}_0$ to the parent region is recorded. For each landmark region $\mathcal{L}_{n+1}$ corresponding to the sub-facets of $\mathcal{F}_0$, child paths are generating by appending $\mathcal{L}_{n+1}$ to the parent's path. For 2D workspaces, an obstacle edge has two vertices and two new paths will be created. For a triangulated 3D surface mesh, three new paths will be created. The process can be repeated by selecting one of the children as the new parent, re-shooting a polytope to the goal, and collecting collisions along this polytope. The selection of children is covered more fully when discussing the region graph search strategy in Section 4.4.1. The polytope-to-goal method is illustrated on a 2D toy problem in Fig. 4-6.

Avoiding Path Loops

If the proposed algorithm uses only the heuristic-based PtG, the approach will not terminate because it could generate path loops in the search tree. This is because the PtG will become trapped in an obstacle surface feature and continuously intersect the same facet. Intuitively, the PtG can be thought of as being trapped in a local minimum where the heuristic cost to the goal is misleading. The challenge to avoiding loops is to reach a new surface where the PtG can once again make progress to the goal. To detect loops, a loop check is performed whenever a new region is added to a path. If a loop is detected, the algorithm attempts to move along the obstacle surface until a point is reached where the PtG can once again make progress to the goal. The process proceeds by generating constraints from neighboring facets. This is illustrated in Fig. 4-7 where red regions are connected as the algorithm moves along the obstacle.

Determining how far to travel along a surface and in what direction is non-trivial. One possibility is moving a certain number of neighbors based on the number of times a sub-facet generates a loop. This approach requires maintaining a visited list for each $\mathcal{L}$. Choosing the direction of travel is also a design parameter. One possibility is to randomize the search direction; another option is to switch directions every time a region is visited.

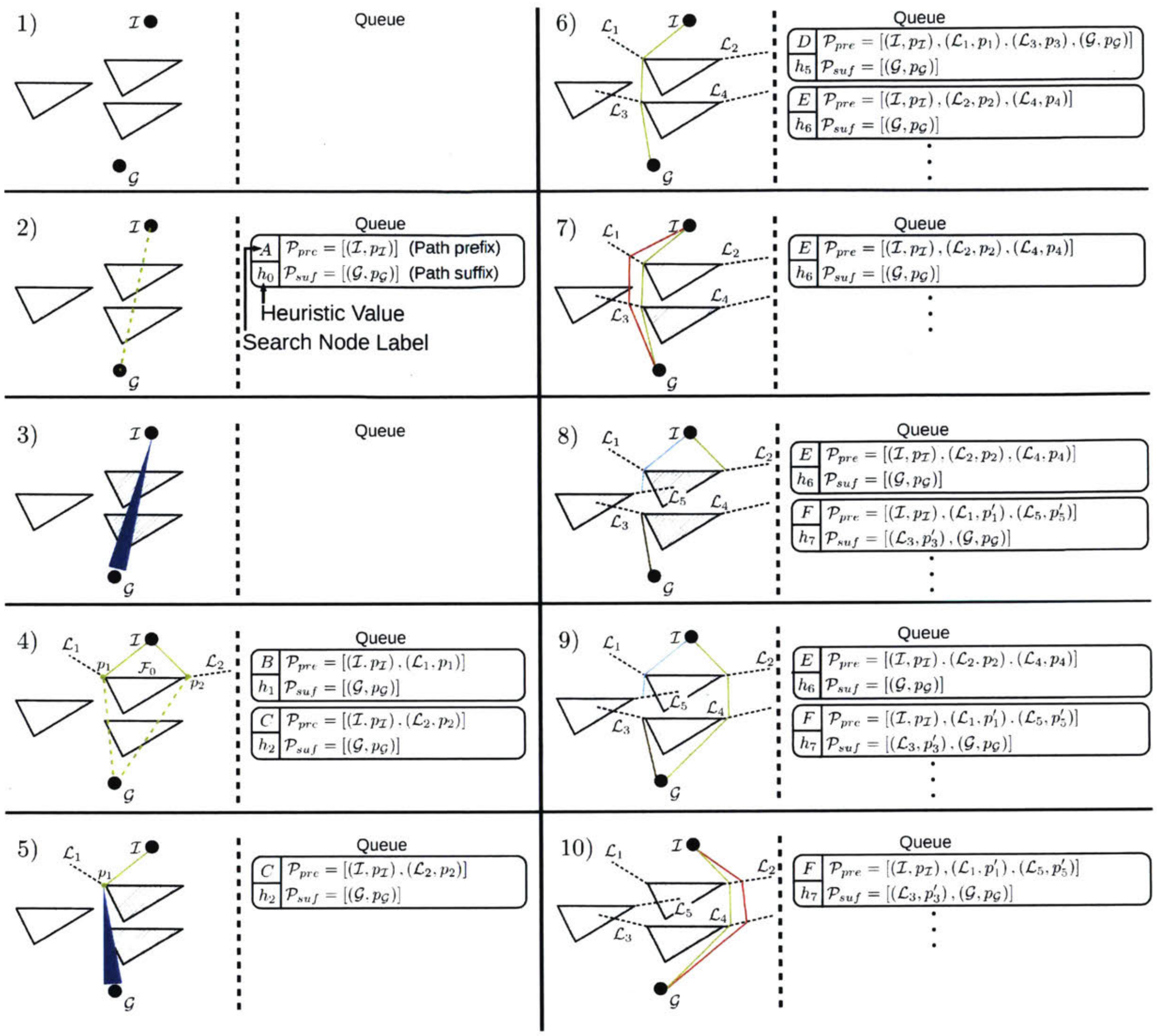
A caveat of using neighboring facets to avoid loops is that the resulting path may be very poor (consider $\mathcal{P} = [\mathcal{L}_1, \dots, \mathcal{L}_7]$ in Fig. 4-7). Because of this, before the region planner passes a path to the trajectory planner, an attempt is made to re-wire it. Specifically, given a path $\mathcal{P} = [\mathcal{I}, \mathcal{L}_1, \dots, \mathcal{L}_{n-1}, \mathcal{G}]$, a subroutine finds a subset of $\mathcal{P}$ that begins at $\mathcal{I}$ and ends at $\mathcal{G}$, which passes collision checks between each $\mathcal{L}_n$. The rewiring is shown in Fig. 4-7 as connecting $\mathcal{I}$ to $\mathcal{L}_5$.

4.4 Searching the Region Graph

The preceding section details the region graph and how it is formed. Regions $\mathcal{R}$ are vertices in the graph and represent constraints that waypoints must satisfy. Regions may be connected using a heuristic-based method or by climbing across polytope facets. Given the form of the vertices and how edges are constructed, it is necessary to establish how the graph is searched. Specifically, it is necessary to specify how the region planner produces paths and how the region planners and trajectory planners interact to produce feasible solutions quickly. A variety of search strategies are possible. For example, a simple strategy is to generate child regions, optimize trajectories that pass through each new child, add the children to a search queue, and continue the process until a terminal child is reached. This strategy perfectly interleaves the region and trajectory planner; every new $\mathcal{R}$ requires a call to the trajectory planner. A downside of this strategy is that trajectory optimization calls are typically computationally expensive and this strategy potentially requires a large number of optimizer calls. This work relies on the observation that many applications require good, feasible solutions quickly rather than provably optimal ones. The idea that many applications require good, feasible solutions quickly rather than provably optimal ones is a common theme in many approaches in robotics and autonomous systems [42]. With this in mind, this work proposes on a hybrid search strategy termed First Feasible Hybrid Search (FFHS).

4.4.1 First Feasible Hybrid Search

First Feasible Hybrid Search has the objective of generating good, feasible paths quickly in complex environments. The key innovation behind FFHS is integrating the region and trajectory planners in a way that accomplishes this task. Because trajectory optimization calls are expensive, the search seeks to minimize them. To do this, trajectory optimizations are performed only on *complete* paths; paths that travel from the initial state and include all goal states. To determine which paths might be good without determining the true path cost via trajectory optimization, path costs are estimated via heuristics. For example, the geometric distance between path regions can serve as an estimate of the path cost. A caveat of this approach is that without performing the full risk-bound trajectory optimization, it is difficult to determine where a trajectory will travel. The risk-bound trajectory will be offset



1)
Queue
2)
Queue
A $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}})]$ (Path prefix)
h_0 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$ (Path suffix)
Heuristic Value
Search Node Label
3)
Queue
4)
Queue
B $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_1, p_1)]$
h_1 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
C $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2)]$
h_2 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
5)
Queue
C $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2)]$
h_2 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
6)
Queue
D $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_1, p_1), (\mathcal{L}_3, p_3), (\mathcal{G}, p_{\mathcal{G}})]$
h_5 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
E $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2), (\mathcal{L}_4, p_4)]$
h_6 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
7)
Queue
E $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2), (\mathcal{L}_4, p_4)]$
h_6 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
8)
Queue
E $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2), (\mathcal{L}_4, p_4)]$
h_6 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
F $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_1, p'_1), (\mathcal{L}_5, p'_5)]$
h_7 $\mathcal{P}_{suf} = [(\mathcal{L}_3, p'_3), (\mathcal{G}, p_{\mathcal{G}})]$
9)
Queue
E $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_2, p_2), (\mathcal{L}_4, p_4)]$
h_6 $\mathcal{P}_{suf} = [(\mathcal{G}, p_{\mathcal{G}})]$
F $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_1, p'_1), (\mathcal{L}_5, p'_5)]$
h_7 $\mathcal{P}_{suf} = [(\mathcal{L}_3, p'_3), (\mathcal{G}, p_{\mathcal{G}})]$
10)
Queue
F $\mathcal{P}_{pre} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{L}_1, p'_1), (\mathcal{L}_5, p'_5)]$
h_7 $\mathcal{P}_{suf} = [(\mathcal{L}_3, p'_3), (\mathcal{G}, p_{\mathcal{G}})]$

Figure 4-8: Illustration of key steps of First Feasible Hybrid Search on a toy problem. 1) The agent must navigate from initial region $\mathcal{I}$. 2) The initial search node A contains a prefix with only the initial region and a suffix with only the goal region; the heuristic cost is the distance between them. 3) A is popped from the queue and a polygon is shot to the goal. 4) Edge $\mathcal{F}_0$ is the closest intersection; two new search nodes are created, B and C, with landmark regions $\mathcal{L}_1$ and $\mathcal{L}_2$ added to the respective prefix paths. Heuristic costs h_1 and h_2 are estimated as $f = g_{pre} + h_{pre,sux} + g_{suf}$ where g_{pre} is the heuristic cost estimate of the prefix path (a solid green line), $h_{pre,sux}$ is the cost estimate from the final prefix region to the initial suffix region (dashed green line), and g_{suf} is the heuristic estimate of the suffix path. 5) Node B has a lower heuristic value, $h_1 < h_2$, and is popped from the queue; a second polytope to the goal is generated from p_1. 6) The prefix path of node D is complete and passed ot the trajectory optimizer. 7) The risk-bound trajectory optimization produces a trajectory (red) that is further from the obstacles than the heuristically-generated path (green). However, it is invalid because it intersects an obstacle unseen to the heuristic-based method. 8) The trajectory is split into a new prefix (cyan) and suffix (brown) based on the conflicting obstacle. The node is re-added to the queue (node F). 9) Node E now has the lowest heuristic value. 10) The trajectory optimized from node E's path is collision free.

from the obstacle surface and may intersect obstacles not seen by the polytope to goal approach described in Section 4.3.3. Because of this, a validation step is required. Therefore, the region planner generates candidate paths and validates them post-trajectory optimization; if a path is invalid (i.e. it intersects an obstacle), landmark regions are added to the path and it is returned to the search queue. First Feasible Hybrid Search is greedy; however, it performs well against optimal search strategies on a number of use cases.

To describe the search, the notions of a *grounded* path, *complete* path, *prefix* path, and *suffix* path are introduced. While a path is a sequence of regions, a grounded path is a sequence of pairs where each pair consists of a region and a heuristic point: $\mathcal{P} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{R}_1, p_1), \ldots, (\mathcal{R}_n, p_n)]$. The heuristic point is a point that lies in the corresponding region. It is used to compute the heuristic cost estimate of the present node in the search tree. Specifically, point p_n is the search node's best estimate of where an optimal trajectory passes through region $\mathcal{R}_n$. A variety of methods are possible to determine p_n. The simplest is to use the point on R_n that minimizes the Euclidean distance between R_n and R_{n-1}. A more computationally demanding method to compute the heuristic points is to perform a scaled-down trajectory optimization. For example, instead of performing a risk-bound trajectory optimization modeling complete nonlinear dynamics, heuristic points may be quickly computed via a convex program. This is the approach discussed in [38]. In addition to serving as heuristic cost estimates, heuristic points serve as the base vertex for the polytope to goal method discussed in Section 4.3.3.

A grounded prefix path is a partial path that begins at $\mathcal{I}$ but does not necessarily contain the final goal region $\mathcal{G}_m$, i.e. $\mathcal{P} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{R}_1, p_1), \ldots, (\mathcal{R}_n, p_n)]$. Similarly, a grounded suffix path is a partial path that ends at the goal region $\mathcal{G}_m$ but does not necessarily begin at $\mathcal{I}$, i.e. $\mathcal{P}_s = [(\mathcal{R}_n, p_n), \ldots, (\mathcal{G}_m, p_{\mathcal{G}_m})]$. A grounded complete path is a path that contains $\mathcal{I}$ and all goals $\mathcal{G}$: $\mathcal{P}_{complete} = [(\mathcal{I}, p_{\mathcal{I}}), (\mathcal{R}_1, p_1), \ldots, (\mathcal{R}_n, p_n), \ldots, (\mathcal{G}_m, p_{\mathcal{G}_m})]$. The goal of the path generator is to connect the *final* element of the current prefix path with the *initial* element of the current suffix path. The region planner starts with prefix path of just $\mathcal{I}$ and a suffix

Algorithm 3: First Feasible Hybrid Search

```
Input: The master problem, Eqns. (2.1) - (2.5)
Output: An open loop controller $\mathbf{u}$ that navigates from $\mathcal{I}$ to $\mathcal{G}$
1  do
2      $Node \leftarrow PathGenerator.NextFeasiblePath()$
3      if $Node.IsComplete()$ then
4          $TrajectoryOptimizer\ (Node)$
5          if $TrajectoryOptimizer.Success$ then
6              $PathGenerator.Validate\ (Node)$
7  while $Node.IsComplete() \wedge \neg Node.IsValidated()$
8  if $Node.IsComplete() \wedge Node.IsValidated()$ then
9      return success
10 else
11     return failure
```

Algorithm 4: PathGenerator

```
Input: The current candidate node $\mathcal{N}$, an environment of polytope obtacles,
       and an initial state and goal state
Output: A complete path $\mathcal{P}$
1  if $\mathcal{N}.IsComplete()$ then
2      $RewirePath\ (\mathcal{N})$
3      return $\mathcal{N}$
4  else
5      $\mathcal{N}_c \leftarrow GenerateNextBestChild\ (\mathcal{N})$
6      $PathGenerator\ (\mathcal{N}_c)$
7  end
```

path of $\mathcal{G}$, i.e. it attempts to connect the initial state to the goal state. In the following, a node in the search tree is denoted $\mathcal{N}$. Each $\mathcal{N}$ consists of a prefix and suffix path, a (possibly optimized) trajectory, and a flag indicating whether the node has been validated.

An overview of the key steps of First Feasible Hybrid Search is depicted in Fig. 4-8. The search and its subroutines are described in Algs. 3 - 6. Algorithm 3 describes the top level hybrid search. The key line is Line 2, where the algorithm attempts to generate a new feasible path given $\mathcal{N}.prefix$ and $\mathcal{N}.suffix$. If a new complete path can be found, the trajectory planner is called using the constraints contained in $\mathcal{N}$ (Line 4). If the trajectory optimization is successful, the path is validated. The algorithm continues while the current node is complete but *not* validated. If the

```
Algorithm 5: GenerateNextBestChild
   Input: A parent search node $\mathcal{N}_p$
   Output: A child search node $\mathcal{N}_c$
 1 if $CheckForLocalMin\,(N_p)$ then
 2     $\mathcal{N}_c \leftarrow ClimbFromLocalMin\,(\mathcal{N}_p.prefix)$
 3 else
 4     $PtG \leftarrow GenPtG\,(\mathcal{N}_{p,prefix}, \mathcal{N}_{p,suffix})$
 5     $ClosestFacet \leftarrow CollisionChecker\,(\mathcal{N}_{p,prefix}, PtG)$
 6     $Children \leftarrow GenChildren\,(ClosestFacet)$
 7     for $\mathcal{N} \in Children$ where $\mathcal{N}.h \neq h_{min}$ do
 8         $Queue.Insert(\mathcal{N})$
 9     end
10     return $\mathcal{N}$ where $\mathcal{N}.h == \mathcal{N}.h_{min}$
11 end
```

current node's path is not complete, this implies the graph generator was unable to find a new complete path and the algorithm fails. For the algorithm to return success in Line 8, the output node must be complete and validated.

4.4.2 Path Generator

Key to the success of Alg. 3 is the ability to efficiently generate complete paths. The pseudocode for this process is given in Alg. 4. *PathGenerator* takes a depth-first strategy to finding good paths. If a search node $\mathcal{N}_p$ is *not* complete, its children are generated (line 5). From the set of generated children, the best child is selected according to a deterministic cost heuristic. Then *PathGenerator* is recursively called. The process repeats until node $\mathcal{N}$ is complete (Line 1).

Children are generated via the two methods described in Sections 4.3.3 and 4.3.3: collisions are collected along a polytope to the goal or, if the search detects a loop, it iterates across neighboring facets. This is described in Alg. 5. The first conditional statement, Line 1, detects heuristic failure via loop checking. If the path does not contain a loop, a polytope is directed to the goal. As mentioned above, the node $\mathcal{N}$ with the smallest heuristic cost h_{min} is returned while all other nodes are inserted into the queue.

Algorithm 6: AvoidPathLoop

```
Input: A prefix path $\mathcal{P}_p$ with a loop.
Output: An updated path that attempts to lead away from the sub-facet
        generating the loop
$MinObject \leftarrow LocalMinVisitedList(\mathcal{P}_{\surd})$
$MinObject.NumVisits \leftarrow MinObject.NumVisits + 1$
$NumNbrs \leftarrow k * MinObject.NumVisits$
$SearchDirection \leftarrow SelectDirection()$
$\boldsymbol{\mathcal{R}} \leftarrow \emptyset$
for $i \leftarrow 0$ to $NumNbrs$ do
    $\mathcal{R} \leftarrow GenNeighbor(\mathcal{R}, SearchDirection)$
    $MinObject.insert(\mathcal{R})$
    $\boldsymbol{\mathcal{R}}.insert(\mathcal{R})$
end
```

4.4.3 Avoiding Path Loops

The algorithm to avoid path loops is described in Alg. 6. A visited list maintains the number of times each sub-facet has generated a loop in the path. This is used for determining how many neighbors to use in order to search around the sub-facet. If it is the first time a sub-facet has generated a loop, the algorithm will add the immediate neighbor to the path and continue the search using the PtG. If the sub-facet has been visited multiple times and generated a loop, the search will move across multiple neighboring facets before restarting its search using the PtG. A search direction is chosen at random. In 2D, there are only two directions in which to climb around an obstacle, clockwise and counterclockwise. In 3D, it is more difficult to define a direction. The implementation of region planner relied on the normal of the constraint surface. If the positive normal of the constraint surface was chosen, the algorithm would climb in a direction indicated by the positive normal. The opposite is true if the negative normal is selected.

4.4.4 Rewiring Complete Paths

An additional optimization regarding the region planner is rewiring the graph after a path has been generated. After exiting the loop avoiding procedure, the new path may be poor because it travels along the obstacle surface. This process is performed without consideration for path cost. An example is shown in Fig. 4-7 where the path could be improved by removing a number of nodes. To prevent poor paths from being passed to the trajectory planner, an attempt is made to rewire complete paths (Line 2 of Alg. 4). Given the complete path $\mathcal{P}' = [\mathcal{I}, \mathcal{R}_1, \ldots, \mathcal{R}_i, \mathcal{R}', \mathcal{R}_{i+1}, \ldots, \mathcal{R}_T, \mathcal{G}]$, the problem consists of finding a subset of $\mathcal{R} \in \mathcal{P}$ that is optimal and minimizes a cost metric. To do this, a simple A* search is used with a collision check before adding neighbors.

4.5 Discussion of the Algorithm's Properties

Soundness: When the state is linear between collocation steps and stochasticity is additive Gaussian noise, the trajectory is probabilistically feasible to within δ, where δ is a small number. This follows from the assumption that the state distribution is known at any future time and a bound on its support can be computed. This bound can be used when calling *IsValid()* and collision checking between collocation steps. A small number δ is necessary because the Gaussian has infinite support; even checking a large region around the path does not mean all probability mass has been captured.

Termination: The algorithm is guaranteed to terminate, either returning a feasible trajectory or failing. This follows from loop checking and the fact that the planning environment contains a finite set of polytopes. A partial path $\mathcal{P}$ with a loop will be detected by *DetectLocalMin*(); if the path enters the same local minimum more than once, extensions to $\mathcal{P}$ will be different because $\mathcal{P}$ is extended based on the number of times the path has visited a local minimum. If $\mathcal{P}$ is still looping after extension (i.e. the algorithm is stuck in a cavern), *NoLoops*() will prevent $\mathcal{P}$ from being added to the queue.

Optimality: First feasible hybrid search is greedy and does not provide a guarantee on finding the optimal solution. The goal of the algorithm is to compute good trajectories quickly. If the use case demands higher quality solutions than the greedy

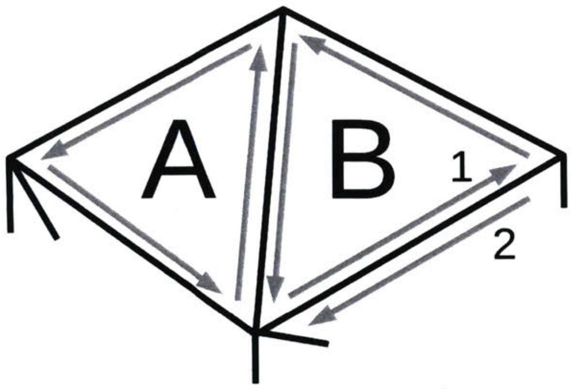


Figure 4-9: Example of half-edge data structure shown in gray for polyhedral facets A and B. Halfedges 1 and 2 illustrate how the algorithm can iterate over neighboring facets. Halfedge 1 has an *opposite* relation that points to halfedge 2 and vice versa. This implementation is inspired by CGAL [61].

solution, it is trivial to make FFHS into an anytime algorithm where the best solution is recorded as an incumbent and the search continues until the queue (Line 8 of Alg. 5) is exhausted. We provide experimental results in Chapter 5 that illustrate FFHS's performance against an interleaved hybrid search that takes a strategy akin to A*, i.e. for every new region added to a path, a trajectory optimization is performed to evaluate the cost of each partial path.

4.6 Computational Geometry Considerations

FOX relies heavily on algorithms & methods from computational geometry. Many of these subroutines (such as collision checks) are called large numbers of times for a path planning problem. It is important that these routines are fast and reliable. Therefore, I include this section to explain some of the details used in FOX's implementation. Specifically, I include descriptions of the geometric data structures and collision checking routines. Overall, components from the Computational Geometry Algorithms Library (CGAL) inspired many of the approaches even if the implementation was custom rather than relying on the library [105].

4.6.1 Geometric Data Structures

Choosing the appropriate data structures to describe objects is often important in computational geometry and this is especially true in the case of FOX. FOX uses the data structure in two ways: collision checking and the *nbr* relation that enables FOX to search over various facets of a polytope. In the 2D case, the primitive describing a polygon is a line segment. Obstacle segments are stored as doubly linked lists so that neighboring edges can be queried. Vertices are also stored and associated with regions $\mathcal{R}$; one vertex has only one region.

In three dimensions, the choices for primitives and data structures are trickier. A three dimensional triangle (a triangle embedded in a three dimensional space) was chosen as the primitive to describe all obstacle surfaces. This simplified the storage and collision checking. A popular data structure to store three dimensional polyhedra is a doubly linked list consisting of facets and half-edges [61]. A half-edge is associated with the edge of a polyhedral facet but is oriented so that queries *next* and *previous* can be defined. Another important query is *opposite*, which returns the oppositely oriented half-edge associated with the neighboring facet. This allows FOX to iterate over neighboring faces.

4.6.2 Collision Checking

Efficient collision checking is a characteristic of any fast motion planning routine. FOX relies on collision queries at various stages; computing the collisions along a polytope to the goal, validating trajectories against collisions, and, to be discussed in a later chapter, checking for collisions against an agent's geometry. Fortunately, because of its relevance to various fields including robotics and video games, many efficient collision checking strategies have been developed [54].

One of the most popular is hierarchical bounding trees. In brief, obstacles in the environment are divided into hierarchical bounding boxes. In this thesis, axis-aligned bounding boxes were used, although many others are possible such as bounding boxes oriented along surfaces. The idea is that checking whether a collision query intersects

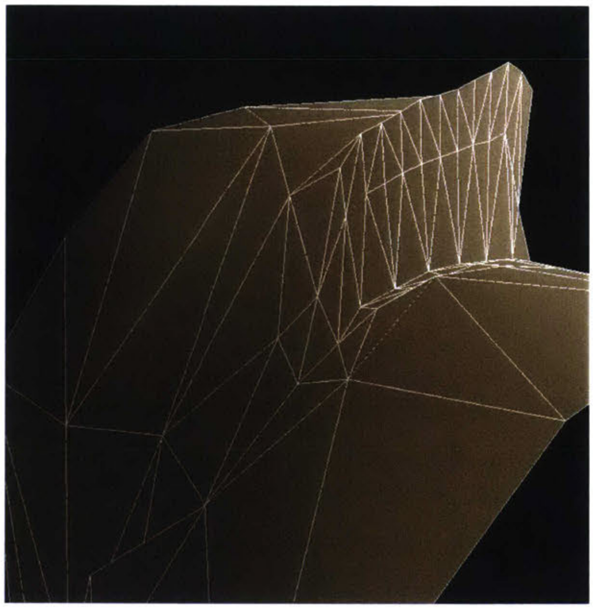


Figure 4-10: Example surface used for motion planning generated from bathymetry data found in [39].

a bounding box is simple and provides an estimate of where a collision may lie. If a collision is detected against a bounding box, the search continues down the tree node representing the colliding box. The process continues until the bounding box is either empty or contains only one or many primitives. At this point, a collision check on the primitives is carried out. Collision checking on primitives is difficult because the algorithms should be efficient as well as robust to numerical precision issues. In the 2D case, my implementation relied on a method for finding whether two segments intersect in [27] and, in the 3D case, my implementation relied on the 3D triangle overlap test described in [40].

4.6.3 Generating the Environments

One of the challenges of motion planning with obstacles is finding a way to generate the data structures that store the obstacles. This is true of any motion planner because of the need to efficiently check collisions or the need to encode the obstacles as constraints. Typically, environment data is captured via sensors with high resolution. If all data points are used to generate obstacles, the environment will become exceedingly complex and slow down the planning algorithm. On the other hand, if too few data points are used, important obstacles information will be lost. Because of these challenges, the problem is non-trivial and an open area of research. FOX uses a number of approaches, depending on the dimension of the workspace.

For some test environments, such as simple 2D workspaces involving trapezoidal obstacles, the obstacles can be randomly generated or, in the case of environments with a very small number of obstacles, hand-coded. For more complex environments, an automated procedure is necessary. A method is discussed in Chapter 5 where the data is downsampled via a method presented in [84] and then the resulting data points triangulated. An example of this for a seafloor environment is shown in Fig. 4-10. Other possibilities include using the Point Cloud Library to generate a surface mesh from a set of points [91].

4.7 Conclusion

This chapter presented FOX, an approach to integrating complex obstacle geometry into the hybrid search's region planner. FOX formulates constraint surfaces and connects them together into paths that can be passed to the trajectory planner. While this chapter focused on FOX, the next chapter focuses on formulating chance constraints using FOX's constraint surfaces. The two methods are combined and results and benchmarks are presented at the end of the next chapter.

Chapter 5

FOX Test Case: Undersea Exploration of the Kolumbo Volcano

This chapter presents a test case that demonstrates FOX in a real world environment. The case investigates the exploration of an undersea volcano, Kolumbo, located off the coast of Greece. Kolumbo is of interest because of its undersea carbon dioxide deposits and hydrothermal vents; monitoring the vents may help predict eruptions [23]. The agent in this scenario is a Slocum glider autonomous underwater vehicle (AUV). Slocum gliders are torpedo-shaped vehicles, which utilize a buoyancy engine to achieve low-energy traversal of the ocean [115]. The buoyancy engine alters the craft's buoyancy so that it moves in a sawtooth pattern through the sea. A set of fins provide lift and produce gliding motion between each peak and trough, giving the glider its name. The buoyancy engine can take various forms; some rely on the temperature gradient of the sea for ultra long-range travel enabling missions in the tens of thousands of kilometers [115]. Other models rely on a ballast pump to change buoyancy and moving an internal battery to affect pitch [103]. The glider and its equations of motion are discussed more in depth in Chapter 7 and Appendix A. For the demonstrations in this chapter, a linear model is assumed between inflections.

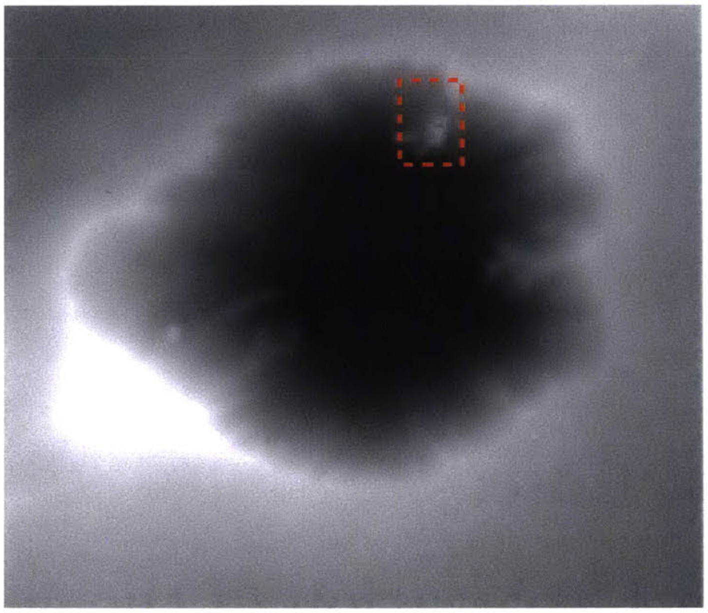


Figure 5-1: 2D map of the Kolumbo Volcano's bathymetry. Darker areas represent deeper regions of the seafloor. One area of interest is a ridge along the northeast corner of the volcano, highlighted with a red outline.

5.1 Problem Statement

FOX's goal is to enable safe, reliable, and dynamically feasible traversal and exploration of the undersea volcano. Each of these points presents unique challenges to FOX, some of which require modification of the base algorithm. FOX should produce trajectories that are safe in that they should avoid obstacles with a margin of safety. That FOX should produce reliable trajectories relates to the uncertainty in the glider's state as a function of the path taken. This relates to the fact that the glider's position state is often poorly known; under the sea, the glider does not have access to a sensor that outputs its global position such as GPS and it must often rely on dead reckoning. This is an important consideration for a motion planner because the planner can take actions that reduce the uncertainty in the glider's position state. For example, rather than take a straight line path, it might be advantageous for the glider to travel near a surface in order to better establish its position relative to the surface using its sonar sensor. Thirdly, FOX should produce trajectories that respect the glider's dynamics. While this chapter considers a linearized glider model, it is still necessary to take into account the glider's sawtooth motion in order to ensure obstacles are avoided. Consideration of pitch and yaw are also important to estimate the glider's energy usage. Finally, FOX should help the glider not only traverse the volcano but also explore various regions. A key task is examining the seafloor around the thermal vents in the caldera's northeast corner.

This chapter proceeds by explaining how FOX can be used to solve various parts of the Kolumbo problem statement. The next section details how FOX's polyhedral model of the environment is automatically generated from bathymetry data. Sections 5.3 and 5.4 detail how the glider's sawtooth pattern is modeled as well as the generation of motion primitives to enable caldera traversal. Sections 5.6 and 5.7 explain how FOX's First Feasible Hybrid Search presented in Chapter 4 can be extended to multiple goals and multiple modes of motion. Finally, Section 5.8 presents the results of various planning cases and benchmarks against other algorithms.

5.2 Seafloor Surface Generation

FOX models the environment using a set of polytopes. For the glider use case, it requires a 3D map of the seafloor obstacles represented as a polygon mesh. Unfortunately, seafloor data is not typically stored as a polygon mesh. Much more likely is that a grid of data points is available, which represent the depth of the seafloor at a specific latitude and longitude. For example, the data provided at [39] represent scans of the seafloor around the Hawaiian Islands at various spatial resolutions from 1 km down to 5 meters.

A simple method to generate a polygon mesh from the data would be to perform a triangulation of the depth data. However, some of the data sets are very large (measuring multiple gigabytes) and generating a mesh from the raw data could generate a large number of spurious polygons. Therefore, an approach is proposed to downsample the data in order to reduce the size of the environments. The approach is inspired by the work in [84], where the focus is on surface generation for image reconstruction rather than path planning.

The method is similar to Principal Component Analysis (PCA). PCA is a method from statistics that, given a noisy dataset, computes a basis for the data that may better explain underlying variables [99]. The insight behind using PCA for surface downsampling is that surface data is often noisy but principal component vectors can explain the surface variation. The method proceeds iteratively by computing the principal components of a subset of surface data and then splitting the subset into finer subsets based on the eigenvalues. Given a subset of data points describing a surface $\mathcal{D}$, i.e. $d_i = (x, y, z)_i$, the covariance matrix is computed as [84]:

$$\boldsymbol{K}_{\mathcal{D}} = \begin{bmatrix} d_0 - \bar{d} \\ \vdots \\ d_i - \bar{d} \end{bmatrix}^T \begin{bmatrix} d_0 - \bar{d} \\ \vdots \\ d_i - \bar{d} \end{bmatrix} \tag{5.1}$$

The eigenvalues of $\boldsymbol{K}_{\mathcal{D}}$, $\{\lambda_0, \lambda_1, \lambda_2\}$, describe the amount of surface variation in the direction of the eigenvectors $\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$. If the minimum surface variation, λ_0, is above some threshold, $\mathcal{D}$ is split along a plane whose normal is equal to the axis of greatest variation, $\mathbf{v}_2$. This process continues recursively: eigenvalues and eigenvectors are computed and the resulting dataset split until λ_0 for each data subset is smaller than the variation threshold. The algorithm is given in Alg. 7. Once the data set $\mathcal{D}$ is downsampled into $\mathcal{D}_{down}$, $\mathcal{D}_{down}$ is triangulated using a Delaunay triangulation to produce the desired polygon surface mesh.

The results of applying this algorithm to the Kolumbo volcano's bathymetry data is shown in Fig. 5-2. The polygons pictured in this image can be used directly for planning.

5.3 Modeling the Glider's Sawtooth Trajectory

Because of the presence of the buoyancy engine, the glider's trajectory follows a unique sawtooth pattern. Under typical operation, the glider moves through the water at a pitch angle of α to the horizontal plane (typical values of α range from 10 to 30 degrees). This idea is illustrated in Fig. 5-3. While it is possible to propel the glider in the horizontal plane, the AUV is unstable and this requires large amount of

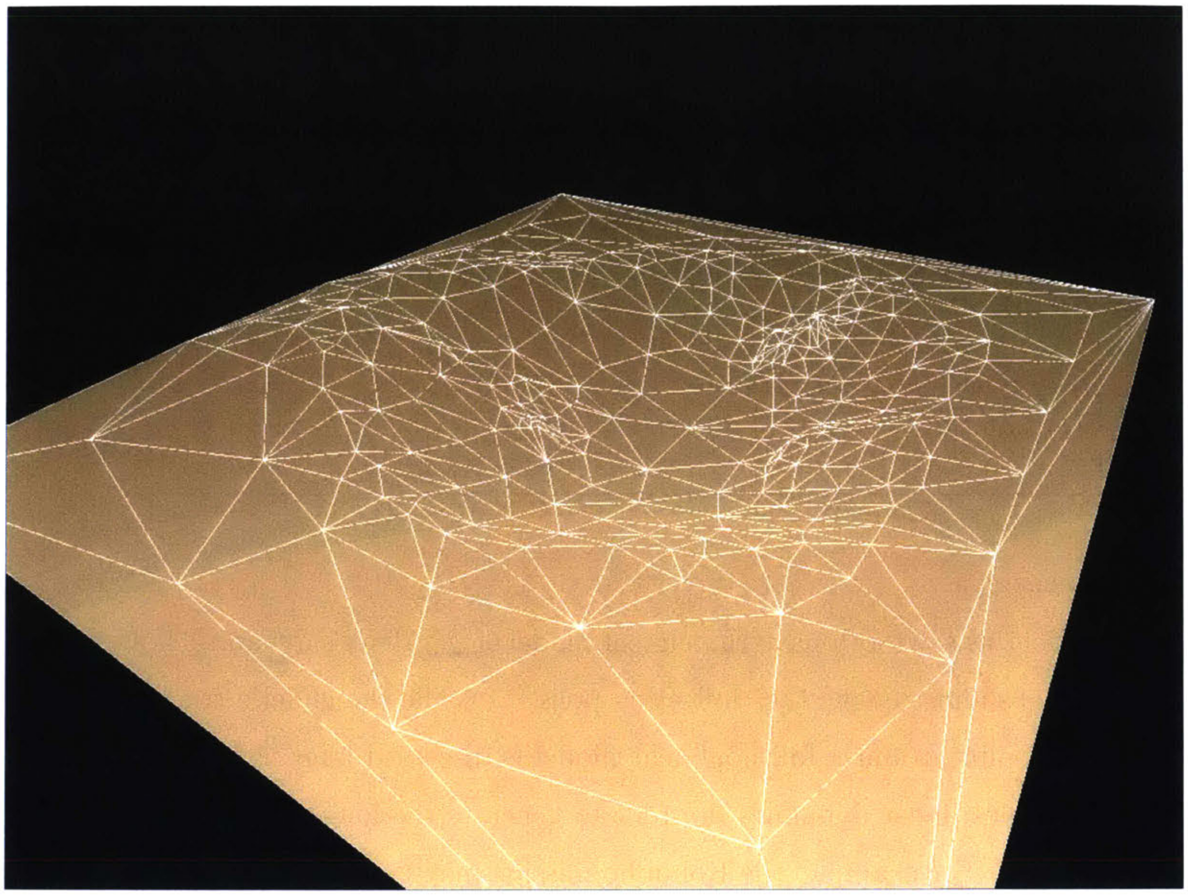


Figure 5-2: The result of applying Alg. 7 to downsample the bathymetry data of the Kolumbo volcano. The undersea volcano is being viewed from the southeast corner.

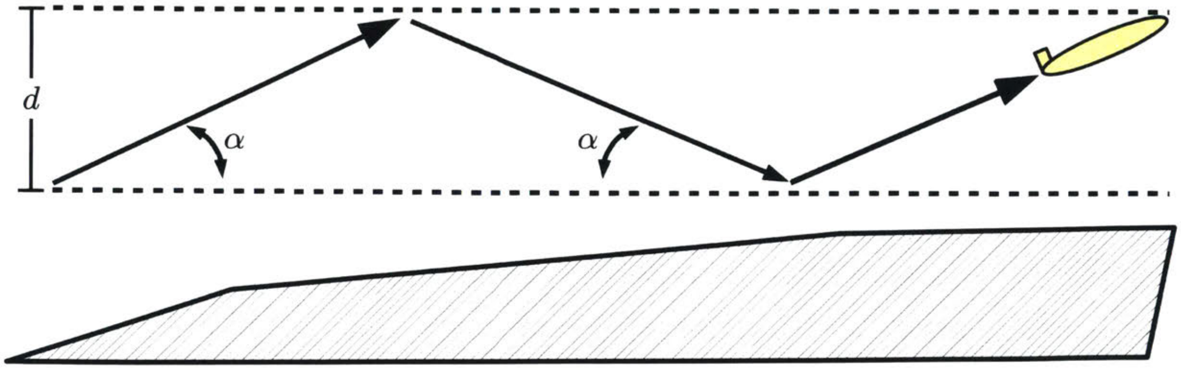


Figure 5-3: Diagram depicting the glider's sawtooth trajectory as it traverses a region of the seafloor. Typically, the glider travels in a depth band of d and moves forward with an angle of α to the horizontal plane.

Algorithm 7: *SeafloorSurfaceGenerator*

```
Input: A dataset $\mathcal{D}$ of points $d_i = (x, y, z)_i$ describing a surface.
Output: A downsampled set of points $\mathcal{D}_{down}$
1  $\boldsymbol{K}_{\mathcal{D}} \leftarrow ComputeCovarMatrix\,(\mathcal{D})$
2  $\boldsymbol{\lambda}, \boldsymbol{v} \leftarrow EigenDecomp\,(\boldsymbol{K}_{\mathcal{D}})$
3  if $\lambda_0 \geq \lambda_{threshold}$ then
4      $\mathcal{D}_1, \mathcal{D}_2 \leftarrow SplitDataset\,(\mathcal{D}, \boldsymbol{v}_2)$
5      $SeafloorSurfaceGenerator\,(\mathcal{D}_1)$
6      $SeafloorSurfaceGenerator\,(\mathcal{D}_2)$
7  else
8      $\boldsymbol{\mu}_i \leftarrow ComputeMean\,(\mathcal{D})$
9      $\mathcal{D}_{down}.add\,(\boldsymbol{\mu}_i)$
10 end
```

actuation energy.

To model this sawtooth pattern, a linear model of the glider's dynamics is assumed between peaks and troughs (i.e. inflection points). While the glider's true dynamics are highly nonlinear and a full nonlinear model is discussed later in this thesis, this chapter assumes linear dynamics in order to generate the long trajectories required by the mission. For example, the Kolumbo mission may require trajectories multiple kilometers in length. Given the linear model, it's necessary to determine placement of the peaks and troughs. The general problem of selecting peak and trough placement along a trajectory is a challenging optimization problem; their location must be selected to minimize energy usage while remaining a safe distance from obstacles. In order to keep trajectory generation fast, a heuristic method was used that quickly generated peak and trough placement via geometry. This was performed using the neighboring landmark regions $\mathcal{L}_n$ and $\mathcal{L}_{n+1}$, assuming a sawtooth width d, and a range of allowable pitch angles α. Given these constraints, a set of cuboid regions was computed that constrained the position of the changes in glider pitch. This construction is shown in Fig. 5-4.

As mentioned above, an advantage of hybrid search is that the regions $\mathcal{L}$ can be arbitrary. In order to constrain the glider's sawtooth trajectory as well as model goal regions, a cuboid region is introduced. Cuboid regions, and all convex regions, are straightforward to convert into constraints on waypoints $\boldsymbol{\omega}$. Waypoints must

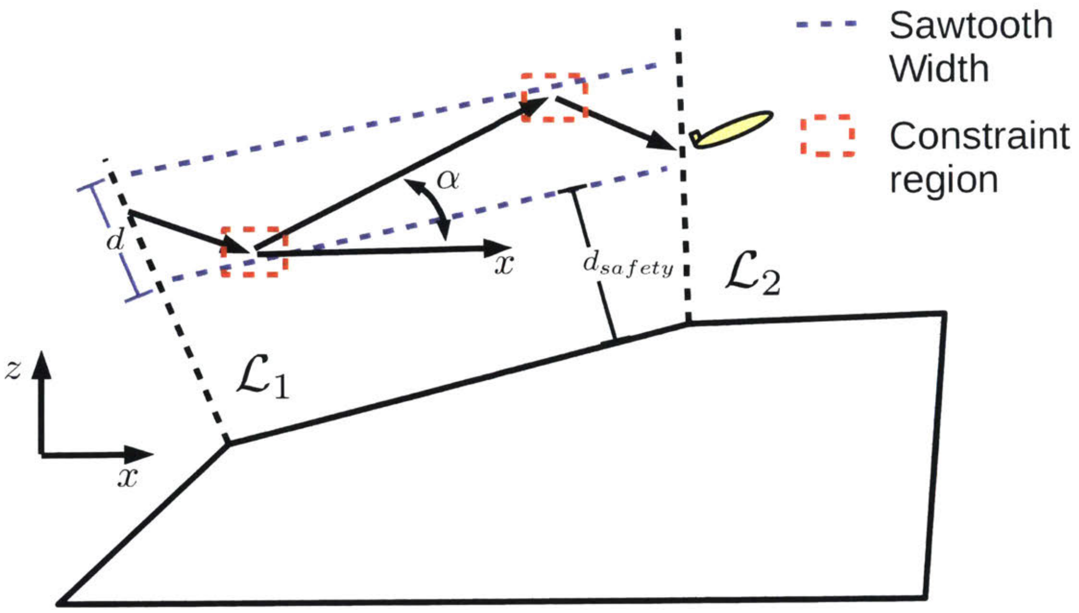


Figure 5-4: Integrating the glider's sawtooth trajectory with FOX's regions $\mathcal{R}$.

simply be constrained to lie on the positive side of a set of planes (or lines in 2D). Specifically, if PL is a set of planes describing a convex region, $\mathcal{L} \in \mathbb{R}^3$, then the waypoint $\boldsymbol{\omega} = (x, y, z)$ must satisfy the constraints:

$$ax + by + cz + d \geq 0 \;\; \forall \;\; pl \in PL \tag{5.2}$$

One final consideration is that the approach described in Fig. 5-3 to generating sawtooth trajectories may be infeasible. Specifically, the glider has a limited maximum pitch angle and, if the slope is too great between two regions $\mathcal{R}$, the glider will be unable to ascend the slope. In these cases, the primitive adopted was to use a doubling back strategy. In this way, the glider would slowly climb at a smaller angle α, doubling back as necessary. This construction is illustrated in Fig. 5-5.

5.4 Constructing Motion Primitives

In addition to the glider's sawtooth motion, the Kolumbo mission requires a number of specific motions to be produced by the planner. For example, exploration of the volcano's northeast ridge requires some form of lawnmower pattern to allow for ade-

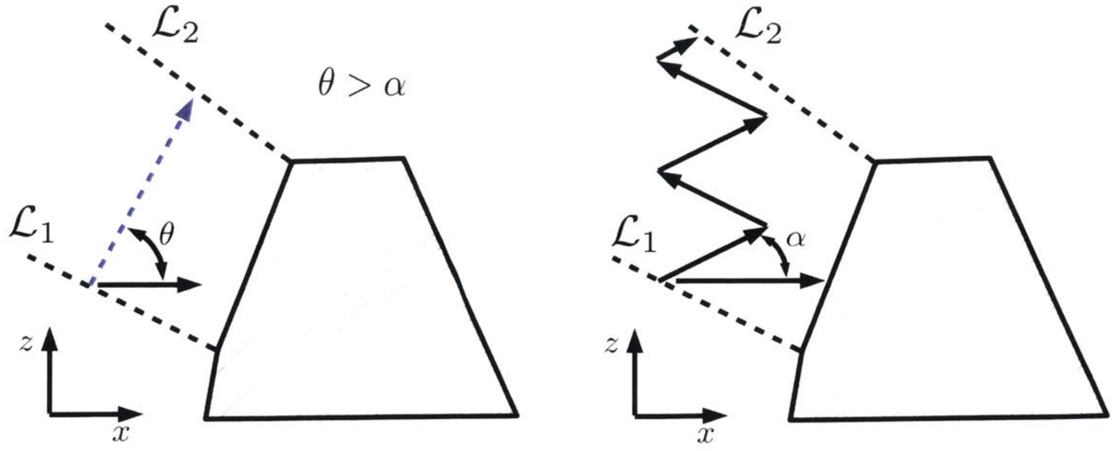


Figure 5-5: If the slope angle θ is too large for the glider's maximum upward pitch, the glider will not be able to attack it straight on; it's necessary to perform a doubling back maneuver.

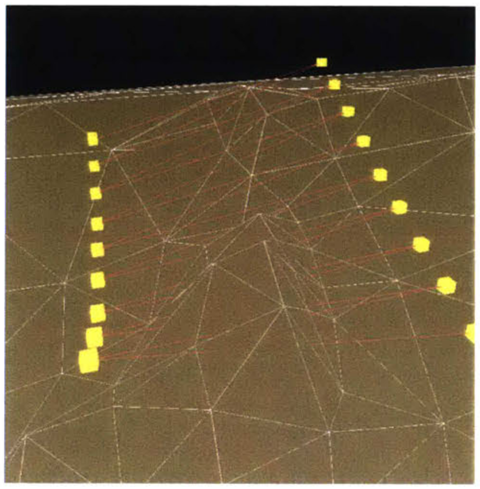


Figure 5-6: An example lawnmower pattern that could be used to explore the ridge at the caldera's northeast corner. The sub-goals of the lawnmower pattern are pictured as yellow cuboids. They are placed on opposite sides of the latitude-longitude bounding box and each is offset a specific distance from the seafloor. The red lines do not illustrate a trajectory; they are drawn to illustrate the ordering of the sub-goals. The glider must travel from the bottom right to the top left of the sub-goals pictured in the image.

quate exploration of the seafloor. These motions are modeled using primitives; their construction is detailed in this section.

5.4.1 Lawnmower Pattern

A lawnmower pattern is used when the glider must explore an area of the seafloor; a lawnmower primitive is deployed to ensure all regions are observed. The pattern is constructed using a bounding box of latitude-longitude coordinates. Two sets of sub-goals are generated on opposite sides of the bounding box. Each goal is offset by a pre-determined distance from the seafloor. In certain cases, the sub-goals may

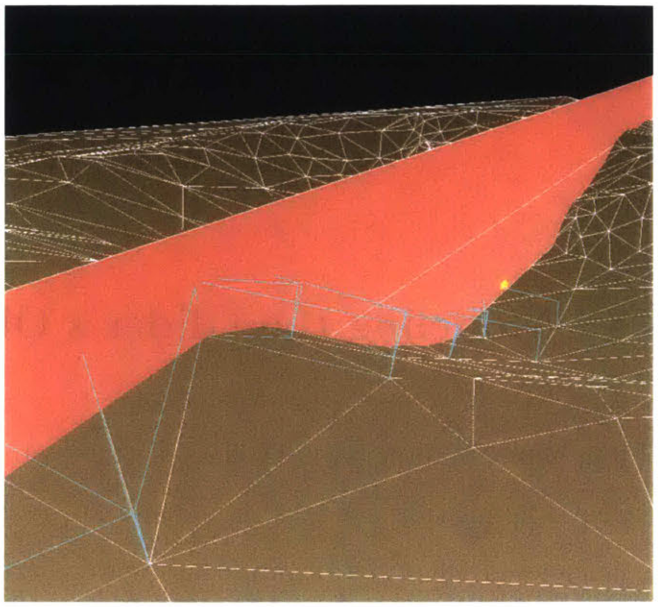


Figure 5-7: An illustration of the method used to construct a path along the seafloor. The red plane indicates the vector traveling from $\mathcal{I}$ to $\mathcal{G}$. The sequence of regions is generated (outlined in cyan) until they reach the polygon under the yellow cuboid goal.

be too close to a steep ridge and difficult for the glider to reach; these sub-goals are removed. The sub-goals are then passed as an ordered set of goals to FOX. Modifying FOX to allow for multiple goal missions is covered in Section 5.6. An example of the sub-goals used in a lawnmower pattern is shown in Fig. 5-6.

5.4.2 Paths Along the Seafloor

As covered in Section 5.7, it may be advantageous to travel along the seafloor to help with localization of the glider. Using FOX's model of the obstacles as a polyhedral mesh, these paths are relatively straightforward to compute. Important is access to each halfedge's neighbors. Recall from Chapter 4 that a halfedge is a directed edge of each polygonal facet; each halfedge has a data structure that points to the halfedge on the neighboring polygonal facet. In a manner very similar to how FOX attempts to climb from local minima, the algorithm generates paths *along* a surface.

These paths are constructed as follows. Assuming FOX must find a path from $\mathcal{I}$ to $\mathcal{G}$ that travels along the seafloor, FOX first computes a vector $\vec{v}_{pc}$ that travels from $\mathcal{I}$ to $\mathcal{G}$. Second, FOX determines the polygon of the seafloor directly underneath $\mathcal{I}$ and $\mathcal{G}$. Let these polygons be $P_{\mathcal{I}}$ and $P_{\mathcal{G}}$. respectively. FOX generates a path from $P_{\mathcal{I}}$ to $P_{\mathcal{G}}$ by starting with $P_{\mathcal{I}}$ as the current polygon P_{cur}. FOX then selects the neighbor of P_{cur} that is in the direction of $\vec{v}_{pc}$. This process is repeated until P_{cur} is the same polygon as $P_{\mathcal{G}}$. An example of this construction is shown in Fig. 5-7.

5.5 Modeling the Glider's Objective Function

Throughout Chapter 4, FOX has assumed objective functions of the path length. Because the mission range is important to the glider, its energy consumption should be considered when performing the hybrid search. To model glider energy usage accurately requires a full nonlinear dynamical model such as that discussed in Chapter 7. To simplify the problem, the test case assumes rules of thumb for energy consumption, i.e. how much energy a certain maneuver will require. There are two important maneuvers to consider: reaching an inflection point of the glider's sawtooth trajectory and changing the glider's yaw. The first maneuver is much more energy intensive because it requires using the buoyancy engine; changing the glider's buoyancy at large depths necessitates operating a pump at high pressures. The second maneuver requires much less energy because one must only change the direction of a small fin at the glider's rear. To model both of these effects and the trajectory's time, a multi-term cost function is assumed:

$$f_0 = c_t \sum_{k=0}^{K} \Delta t_k + c_{inflection} \sum_{k=0}^{K} \mathbb{1}_{inflection,k} + c_{yaw} \sum_{k=0}^{K-1} \Delta_{yaw,k,k+1} \tag{5.3}$$

The first term represents a penalty on the trajectory's time; c_t is a constant that penalizes time usage and Δt_k is the time taken by each trajectory knot. The second term relates to the energy usage per inflection; $c_{inflection}$ represents an estimate of the energy used by an inflection and the indicator function $\mathbb{1}_{inflection,k}$ represents whether there is an inflection at knot point k. The final term relates to the energy usage by changes in yaw; c_{yaw} is the estimated energy usage per change in degrees of yaw and $\Delta_{yaw,k,k+1}$ represents the change in glider yaw from trajectory knot k to $k+1$.

5.6 Enabling FOX to Handle Multiple Goals

A characteristic of the undersea mission is that the motion planner must be able to handle multiple goals. There are two reasons for this. First, missions are often framed in terms of multiple goals. For example, the glider may be dropped off at one location, be required to accomplish a science mission along the seafloor, and return to the drop off point. Second, the glider may need to accomplish motion primitives that consist of multiple sub-goals. For example, the glider may need to perform the lawnmower pattern described in Subsection 5.4.1.

While the approach presented throughout Chapter 4 focused on single initial state and single goal problems, it may be extended to multiple goals. In order to reduce the complexity of the search problem, this use case takes a strategy similar to ScottyPath and ScottyActivity as presented in [38]. The activity planning problem (i.e. the problem of finding an ordering of goals) and the motion planning problem (how to travel from one goal to the next) are separated into solving two different problems. To do this, I introduce a simple activity planning algorithm similar to ScottyActivity. Given a set of goals $\mathcal{G}$, the purpose of this activity planner is to output a sequence of goals which minimizes a cost estimate of the mission. Where path has been used to denote a set of regions traveling from an initial state to a goal state, in the multi-goal case, the definition is broadened to mean a set of regions between an initial state and a goal state *or* between two goal states. A mission is used to refer to a set of paths $\mathcal{P}$ where each path connects a pair of goals (or the initial state to a goal) in $\mathcal{G}$.

Given a set of goals, all of which must be achieved, the problem can be reduced to a traveling salesman problem (TSP). TSPs are NP-Hard problems to solve, although a large amount of effort has gone into effective algorithms to solve them [90]. The approach used in this use case to solve the TSP and order the goals relies on a simple branch and bound strategy. This is reasonable under the assumption that there is likely to be a relatively small number of goals. In order to solve the TSP via branch and bound, there must be an estimate of the cost to travel between goals. This is computed via a simple heuristic; the distance between each goal. This is a downside to

Algorithm 8: *PathGeneratorMultGoal* for the multi-goal case.

```
Input: The current candidate node $\mathcal{N}$, an environment of polytope obtacles,
       an initial state, and a sequence of goal states $\boldsymbol{\mathcal{G}}$.
Output: A set of paths $\boldsymbol{\mathcal{P}}$ that satisfy all mission goals, or failure
1  if $\mathcal{N}.IsMissionComplete()$ then
2      $RewirePath\,(\mathcal{N})$
3      return $\mathcal{N}$
4  if $\mathcal{N}.IsPathComplete()$ then
5      $\mathcal{G}_j \leftarrow SelectNextUnassignedGoal(N)$
6      $\mathcal{N}_c \leftarrow InitNodeStartingNextPath\,(\mathcal{N}, \mathcal{G}_j)$
7      $PathGeneratorMultGoal\,(\mathcal{N}_c)$
8  else
9      $\mathcal{N}_c \leftarrow GenerateNextBestChild\,(\mathcal{N})$
10     $PathGeneratorMultGoal\,(\mathcal{N}_c)$
11 end
```

separating the activity and motion planning; the heuristic may be inaccurate because of obstacles between pairs of goals. However, it allows for a fast algorithm.

The algorithm *PathGenerator*, Alg. 4, is modified slightly to allow for multiple goals; this is presented in Alg. 8. This algorithm receives the same input as the original *PathGenerator* other than accepting a sequence of goals, $\boldsymbol{\mathcal{G}}$. The sequence is the output of the branch & bound procedure described in the previous paragraph. The main difference between Alg. 8 and the single goal case is that Alg. 8 has three branches. If the mission is complete, the current node is returned; similarly, if the current path is not complete because the current goal is not reached, *GenerateNextBestChild* is called, similar to the single goal case. A new branch is added and handles the case if the current path is complete but the mission is not, Line 4. In this case, a new node is generated with the next unassigned goal and the algorithm is called recursively.

5.7 Constraining Variance - FOX with Trajectory Modes

A unique characteristic of this use case is that the shortest trajectories may not always be the best. This is due to the glider's poor localization ability when traveling in open water. Under the sea and unable to establish its absolute position via GPS, the glider must rely on a set of sensors that measure its position relative to the seafloor and sea surface and its velocity relative to the ocean current. An additional challenge is that the sensors may only be able to deliver reliable readings when the glider is within a certain distance to the seafloor. For example, a test glider at the Woods Hole Oceanographic Institute is equipped with a sonar sensor which is able to rotate around the glider's longitudinal axis and measure distances to the seafloor and sea surface. Unfortunately, the sensor is only reliable at a distance of approximately 50 meters from the obstacle surface. If the glider is more than this distance away from the surface, it must rely largely on dead reckoning for its position state estimate. This can be highly inaccurate because of fast moving ocean currents.

To alleviate this issue, the motion planning problem posed in Chapter 4 is modified slightly. Rather than search only for shortest paths, it's necessary to trade off path utility (i.e. distance or energy usage) with the increased uncertainty that a candidate path may generate. This trade-off is modeled via a constraint on the uncertainty in the glider's position over the course of a path. This constraint is referred to as a variance constraint. The variance constraint is different than the chance constraint discussed throughout this thesis. Whereas the chance constraint models the probability of mission failure, the variance constraint bounds the agent's position state uncertainty. In other words, most examples investigated in this thesis assume the underlying stochastic process describing the agent's state distribution does not change. Typically, the variance of the state distribution increases as time increases. In this case, it is possible to reduce the state distribution's variance depending on the agent's trajectory.

To model the increase in variance over the course of a trajectory, a simple linear

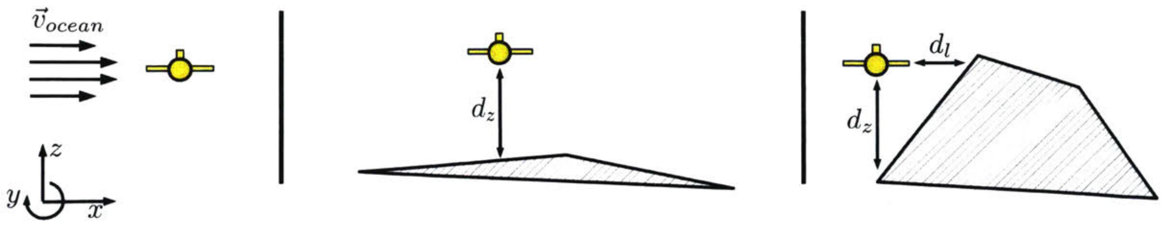


Figure 5-8: Illustration depicting three trajectory modes and their relation to localization using the glider's distance sensors. In all three images, the glider is traveling along the $\hat{y}$-axis. In the left pane, the glider is traveling through open ocean; it is too far from the seafloor for its sensors to produce reliable readings and it must rely largely on dead reckoning. This is difficult because ocean currents can take it off course. The middle pane shows the glider traveling over the seafloor; in this mode, the glider can reliably sense its one dimensional distance d_z to the seafloor. From a localization perspective, the ideal situation is when the glider travels along a ridge. In this case, it can track its height off the seafloor and lateral distance to the ridge, d_l.

model is used. As the glider travels a trajectory, the variance changes at a rate of $c_m t$, where c_m is a constant that denotes the change in variance per unit time for trajectory mode m. To model situations where variance decreases along a trajectory (i.e. due to better sensor measurements), it is possible that c_m is negative. This is in contrast to the stochastic processes considered previously in this thesis, where state variance increases with time. The trajectory modes are properties of the agent and its sensor model. In the glider test case, three modes are assumed. The first mode is traveling through open ocean when distance sensors are out of range of the seafloor. Due to the need to rely heavily on dead reckoning, this has the highest increase in variance per unit time, i.e. the largest c_m. The second mode is traveling along the seafloor in which the one dimensional distance from the seafloor is assumed known due to the presence of a distance sensor. The final mode is traveling parallel to a gently sloping seafloor. This mode incurs the lowest increase in variance because it is assumed two sensors are able to deliver reliable information: a sensor measuring the distance from the glider's bottom to the seafloor and a sensor that can measure the lateral distance to the seafloor. The three modes are illustrated in Fig 5-8.

The search over modes is integrated into the hybrid search as follows. Similar to

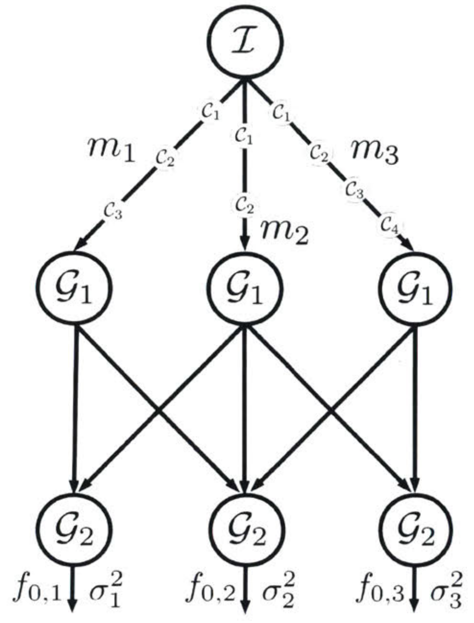


Figure 5-9: Example search tree for determining which mode to use for each path. Nodes record both their estimated cost to reach the goal f_0 and their estimated variance σ^2. The search tree branches on each mode m and an estimate of the increase in variance is computed. If the variance associated with mode m exceeds σ^2_{max}, the node is pruned. While it is possible to switch modes between regions $\mathcal{R}$, to simply the search, I assume that the mode remains that same for an entire path. For example, if the glider is traveling from $\mathcal{G}_1$ to $\mathcal{G}_2$ and it is in mode "bottom following," then it will attempt the entire path in bottom following mode rather than switching.

the single mode case, the algorithm is given a sequence of goals $\boldsymbol{\mathcal{G}}$ and attempts to construct a trajectory that connects all goals. Also input to the problem is a maximum variance σ^2_{max}; any feasible solution must have a smaller estimated variance than σ^2_{max}. To accomplish this, not only do search nodes in the tree maintain a current estimate of their cost but also their position state variance, σ^2. If any path reaches a larger variance than σ^2_{max}, it is pruned. The algorithm for *PathGeneratorMultiGoal* must only be modified slightly to allow search over multiple modes. Rather than attempting to complete a path for the next unassigned goal, as in Alg. 8, the algorithm iterates over each mode and estimates the increase in variance. The search takes a greedy strategy; the node with the lowest cost whose estimated variance does not violate σ^2_{max} is selected for expansion. The search tree is illustrated in Fig. 5-9.

5.8 Results

A number of test cases were computed as well as a series of benchmarks against other algorithms.

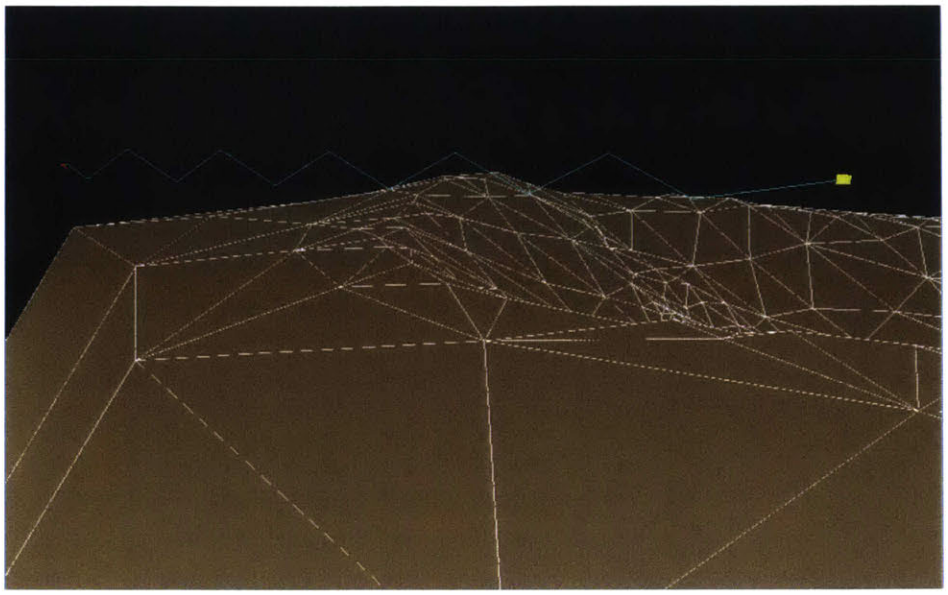


Figure 5-10: Example of a simple glider trajectory that illustrates its sawtooth pattern. The glider travels from the red trapezoid to the yellow cuboid at the caldera's center. While the depth of the initial state and goal is the same; the glider travels in a sawtooth pattern rather than straight line due to dynamical stability and energy usage. As mentioned in Section 5.7, it's unlikely the glider would utilize such a trajectory through the open sea due to difficulty localizing.

5.8.1 Test Cases

Figures 5-10 - 5-13 present FOX test trajectories. Fig. 5-10 shows a simple example of the sawtooth pattern trajectory for a simple case where the glider simply must navigate in the open ocean while remaining in the same depth band. Fig. 5-11 presents an example of the glider traveling along the seafloor in order to reduce its localization variance. Another maneuver to further reduce the position uncertainty is to travel around the caldera; an example of this trajectory is shown in Fig. 5-12. Finally, an example of the lawn mower pattern exploring the ridge in the caldera's northeast corridor is shown in Fig. 5-13.

5.8.2 Benchmarks

The First Feasible Hybrid Search (FFHS) approach was benchmarked against two other approaches. First, the approach was benchmarked against the popular Rapidly-Exploring Random Tree (RRT) algorithm [69]. In brief, the RRT algorithm draws

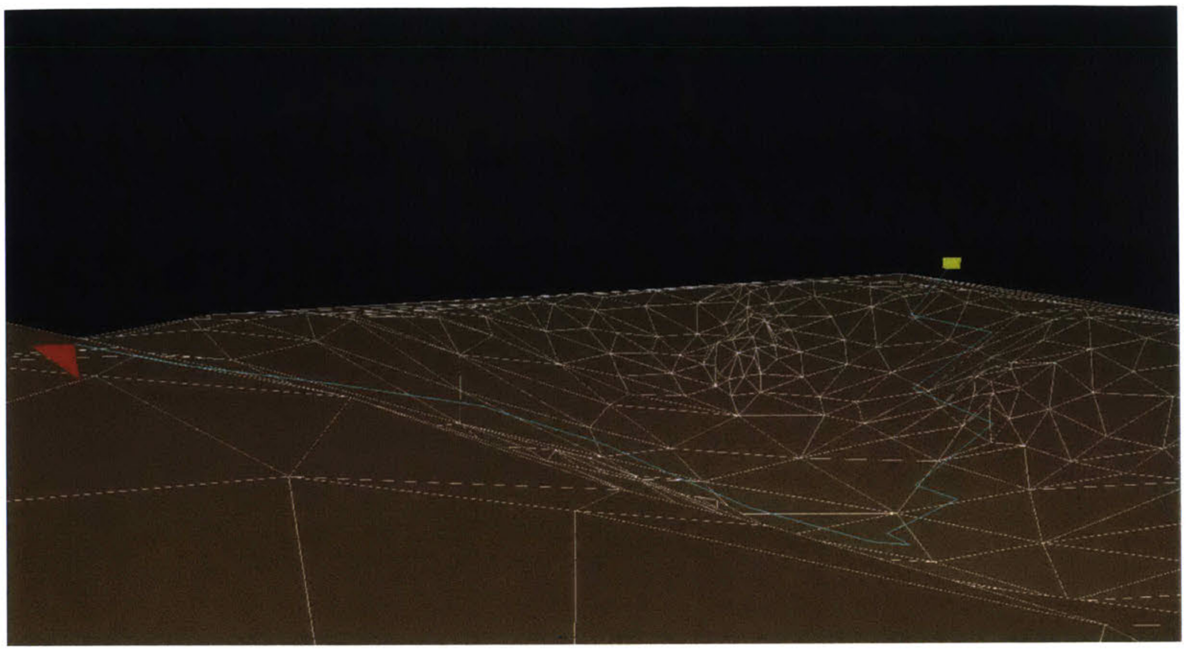


Figure 5-11: Rather than taking a straight line trajectory through open seas and risk incurring uncertainty in the glider's position as in Fig. 5-10, a better trajectory is to travel along the seafloor to allow localization with respect to the distance from the seafloor. In order to travel from the seafloor to the goal, which lies at a shallow depth in the center of the caldera, the glider uses a doubling back strategy to move closer to the surface while remaining at roughly the same latitude and longitude.

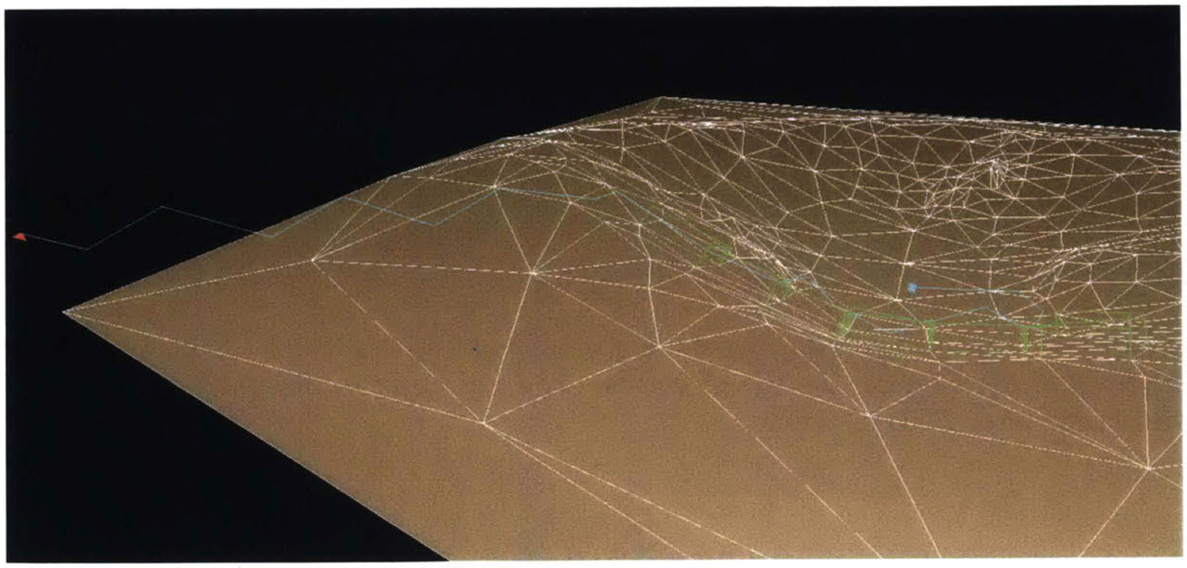


Figure 5-12: Example trajectory of the glider traveling around the volcano's caldera in order to reduce uncertainty about its position. The glider begins at the red tetrahedron on the left side of the image and travels in a sawtooth pattern until it reaches the caldera. The goal is the cyan cuboid at the caldera's center.

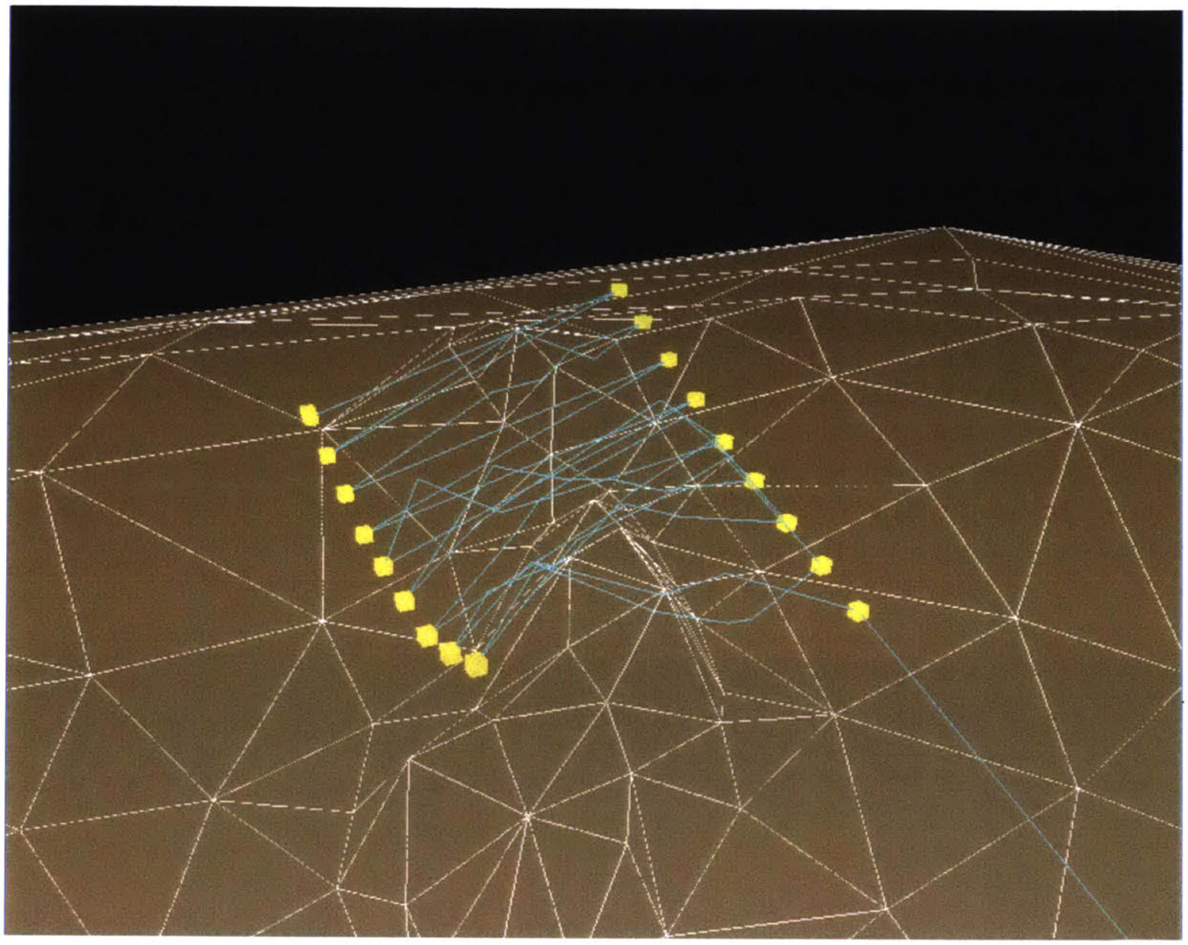


Figure 5-13: Illustration of the lawnmower motion primitive where the glider explores the ridge at the caldera's northeast corner. The trajectory is denoted by cyan lines with lawnmower goals in yellow.

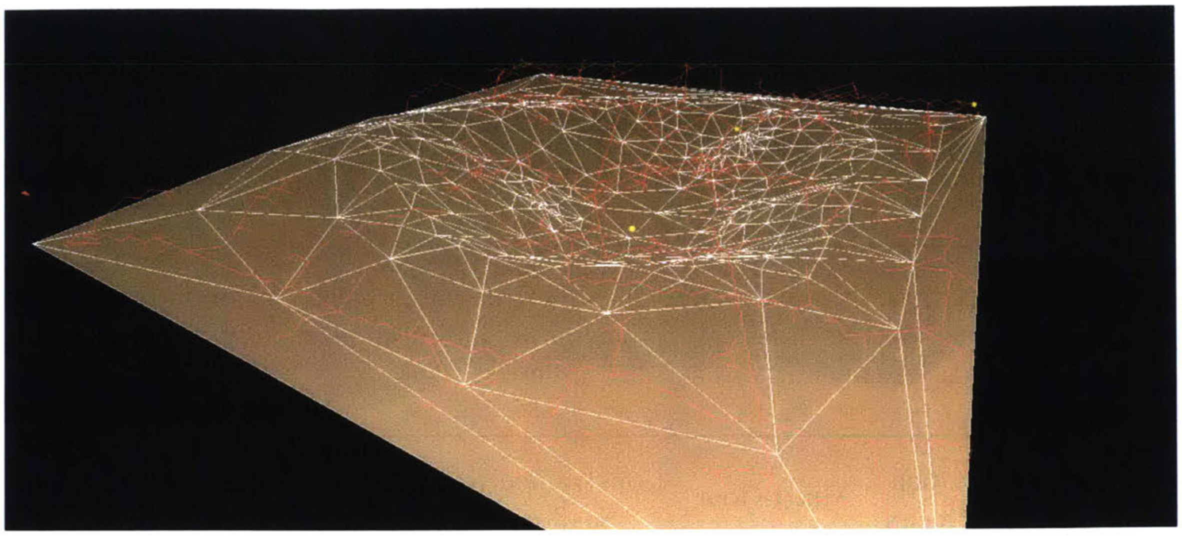


Figure 5-14: Example trajectory tree generated by the RRT algorithm for the multi-goal test case. The trajectory is the red line; goals are the yellow cuboids.

samples from the agent's state space, computes a set of nearest neighboring points along a trajectory that are known to be feasible, and determines if it can extend each neighbor to the new sampled point. To generate the glider's sawtooth trajectories using sampling, rather than extend existing points to the new neighbors via a straight line, extensions were generated in the direction of the new sampled points that were at a feasible glider pitch angle relative to the vertical axis. Extensions were incrementally generated in the direction of the sampled points until they were either infeasible (due to an obstacle collision) or the extended trajectory was at the same latitude-longitude position as the sampled point. While RRT is good at exploring the search space through sampling, it must be modified to account for constrained trajectories such as those that must travel near a surface as described in Section 5.7. One possibility is to sample points only from the region of interest. To perform the benchmarks described here, the sampling was unmodified but paths that did not travel within a certain distance to a surface for a pre-specified amount of time were pruned. This meant that the RRT algorithm would only succeed once it had produced a trajectory that not only reached the goal but also traveled close to the seafloor.

A second benchmark was performed against a flavor of hybrid search. As discussed in Chapter 2, one of the characteristics of hybrid search is that there are many ways

	Open Ocean	Multi-Goal	Constrained to Seafloor	Lawnmower Pattern Small	Lawnmower Pattern Large
FFHS	21	273	102	152	1041
RRT	41	531	582	966	2533
Interleaved	20	19300	8100	36200	Timeout

Table 5.1: Comparison of search times for different search strategies on various test cases. Times listed are the times taken to generate an initial feasible solution. Times are in milliseconds. A timeout was set at 5 minutes.

	Open Ocean	Multi-Goal	Constrained to Seafloor	Lawnmower Pattern Small	Lawnmower Pattern Large
FFHS	31.4	337	174	356	1041
RRT	31.3	2629	1732	669	24750
Interleaved	31.4	321	170	321	Timeout

Table 5.2: Comparison of the value of the cost function, Eq. (5.3), for test cases. Values listed are the objective value for the initial feasible solution. A timeout was set at 5 minutes.

in which the path planner and trajectory planners can interact. One possibility is that the trajectory planner is called after every successor is generated by the region planner. This is similar to the strategy used by ScottyPath in [38]. This is termed an *interleaved* hybrid search because of how the region planner and trajectory planners are tightly interleaved. This contrasts to the depth first strategy of First Feasible Hybrid Search.

FFHS was benchmarked against the RRT and Interleaved Hybrid Search ("Interleaved") and results are presented in Tables 5.1 and 5.2. The "Open Ocean" test case refers to the glider traveling through open ocean on a relatively simple mission, as pictured in Fig. 5-10. The "Multi-Goal" test case involves spacing 4 goals throughout the Kolumbo caldera. "Constrained to Seafloor" is a test case where the glider is required to maintain a bound on its position variance by searching for paths along the seafloor, similar to Fig. 5-11. Finally, two tests were performed involving a lawnmower pattern. "Lawnmower Pattern Small" refers to a small lawnmower pattern with only 5 goals. "Lawnmower Pattern Large" refers to a lawnmower pattern with

21 goals. This is pictured in Fig. 5-13.

All methods perform well on the "Open Ocean" test case. This case is straightforward as the glider does not need to avoid obstacles and must only produce a sawtooth trajectory. The advantage of performing FFHS versus the Interleaved Hybrid Search is seen on the search times for the three complex test cases; search times of FFHS can be measured in milliseconds while search times for Interleaved Hybrid Search are better measured in seconds; on the test "Lawnmower Pattern Large," the interleaved search does not reach a solution before a 5 minute timeout. This is because environments can be very complex and contain a huge number of implied regions. Furthermore, the number of paths is exponential in the number of regions; there are a huge number of potential paths that could be passed to the trajectory planner. While trajectory planner calls are typically measured in hundreds of milliseconds, calling the trajectory planner hundreds of times over the course of one planning instance can slow the overall search. The insight behind FFHS is that only a small subset of the paths should be optimized by the trajectory planner.

FFHS compares well against RRT in terms of time to find a solution and trajectory cost value. One advantage that FFHS has over the RRT implementation used in these tests is the use of motion primitives. FFHS relies on a motion primitive to compute the path along the seafloor; this is quicker to generate than the modification of RRT that simply prunes paths that do not travel along the seafloor. It is likely possible to generate a better sampling scheme for RRT, such as only sampling points from regions close to obstacles. Furthermore, while FFHS compares well in terms of objective value, one advantage of RRT is that it can be extended such that sampling continues once a feasible solution is found in order to improve the objective of the incumbent solution. An example of this is the popular RRT* algorithm [56]. As would be expected, when the Interleaved Hybrid Search produces a solution, it is generally of better quality than FFHS. This is because FFHS takes a greedy approach to the hybrid search to improve its speed. In general, the Interleaved Hybrid Search's solutions are less than a 10% improvement over the FFHS solutions. Whether this improvement in solution quality is worth the wait is up to the user; it may be the case that the user has a long time to generate plans before execution. On the other hand, in cases where the user may want to test a large number of missions to determine which best suits their needs, the time to generate a trajectory may be a more desirable property.

5.9 Conclusion

The purpose of this chapter was to present an application of the FOX algorithm to a test case with real world applicability. FOX succeeds at satisfying the problem statement by providing safe, reliable trajectories that respect the glider's motion and allow exploration of the volcano. FOX provides safe trajectories via obstacle avoidance. FOX allows for reliable generation of glider motion plans via considering different trajectory modes and an estimate of the change in localization error associated with each. By integrating the sawtooth trajectory pattern into FOX, the FOX is able to model the glider's inflection points and their energy usage. Finally, FOX enables volcano exploration through the use of multiple goals and different motion primitives such as a lawnmower pattern. These results are established through various tests and benchmarks against other approaches.

Chapter 6

FOX with CDF-based Probabilistic Constraints

The probabilistic (or chance) constraint bounds the probability of mission failure. As mentioned in Chapter 2, the general constraint is intractable to compute and some form of simplification is required with respect to the distribution or integration bounds. This chapter focuses on chance constraints where the approximation uses a cumulative distribution function (CDF) to evaluate the constraint. This enables an easy-to-compute approximation of the probability of failure. In many cases, the resulting probabilistic constraint is convex, which means it can be used in efficient convex optimization routines. However, using a CDF-based constraint often requires many assumptions in terms of the obstacles' geometry, agent's geometry, agent's dynamics, and stochasticity. Following chapters focus on more complex models of uncertainty.

The chapter proceeds as follows. First, a background on CDF-based chance constraints is provided. Second, I build on prior work to show that many cases of this constraint are convex. Next, I explain how the CDF-based chance constraint is integrated into the FOX algorithm, which was presented in the previous chapter. Finally, I provide benchmarks and results the show the effectiveness of FOX as a risk-aware path planner.

6.1 CDF-Based Chance Constraint Approximations

The chance constraint models the probability of mission failure, e.g. the probability of colliding with an obstacle. The general case is difficult to compute because it involves the integral of a probability distribution over possibly many dimensions as the state evolves in time. The general form of the discrete-time chance constraint is:

$$P\left(\bigwedge_t \mathbf{x}_t \in \mathcal{W}_{free}\right) \geq 1 - \epsilon \qquad (6.1)$$

where, in motion planning problems, $\mathcal{W}_{free}$ is some obstacle-free region of the workspace and ϵ is a failure threshold. Eq. (6.1) may be referred to as a *joint* chance constraint because it constrains the probability of failure at all time steps in a discrete time stochastic process. Many methods to simplify the probabilistic constraint have been discussed in literature [80], [12], [83]. While much work has focused on numerically approximating the constraint using sampling, other work, including the approach taken in this chapter, approximates the constraint by computing the probability of failure using some form of the underlying distribution's cumulative distribution function (CDF).

Using a CDF-based approximation has a number of advantages. First, it is simple to evaluate. Assuming that the uncertainty is Gaussian, the joint chance constraint, Eq. 6.1, can be approximated by evaluations of a single variate or multivariate Gaussian CDF at each time step. The single variate Gaussian CDF is accessible in various numerical libraries; a multivariate version is available via [36]. Second, as shown in the following section, the constraint is convex in a large number of cases, which makes it amenable to convex programming.

On the other hand, using a CDF approximation requires assumptions. First, assumptions must be made about the stochastic model in order to separate the joint chance constraint into individual chance constraints. Second, assumptions must be made regarding obstacle and agent geometries. Obstacle geometry enters the problem as limits of integration; it is typically simplified in order to allow for tractable limits of integration. Additionally, the agent's geometry is often ignored; all its geometry is assumed concentrated at a single point. Finally, the assumption of discrete time Gaussian uncertainty is inaccurate for real world continuous time and often nonlinear systems. Many of these assumptions will be dealt with in later chapters. This chapter focuses on CDF-based methods as a simple, fast-to-evaluate method to model the probability of collision.

The first simplification for CDF-based methods is to make an assumption about the stochastic process' interdependence at different time steps. This allows the intractable joint chance constraint to be broken into simpler individual chance constraints. As mentioned as a benchmark in Chapter 3, an innovative approximation uses Boole's inequality. The approach relies on Boole's inequality to derive a conservative approximation of the joint chance constraint:

$$P\left(\bigvee_i A_i\right) \leq \sum_i P\left(A_i\right) \tag{6.2}$$

Computing a trajectory that satisfies the RHS of Eq. (6.2) implies that the joint chance constraint is satisfied. Because the RHS of Eq. (6.2) sets all probability terms with overlapping A_i to zero, it assumes that the events are mutually exclusive. By setting these terms to zero, the RHS of Eq. (6.2) overestimates trajectory risk and is therefore conservative with respect to the risk if one were to compute the joint chance constraint.

Another method to approximate the joint chance constraint is through independence; under independence assumptions the probability of trajectory success (failure) is the product of the individual probabilities. Similar models have been discussed in other work such as [108]; to the best of the author's knowledge, this work is the first time the formulation is used within a convex path planner.

In the formulation used in this thesis, the probability of success at time $t = T$ is modeled by conditioning on the success of past transitions and the law of total probability:

$$P(S_t) = P(S_t|S_{t-1}, \ldots, S_1)\, P(S_{t-1}|S_{t-2}, ..., S_1) \ldots P(S_1) \tag{6.3}$$

where S_t is the event of the agent succeeding at time step t. Each term is assumed Markovian: the probability of success at time t is conditionally independent of success at $t-2$ given its success at $t-1$. Using this fact, (6.3) can be simplified:

$$P(S_T) = P(S_t|S_{t-1})\, P(S_{t-1}|S_{t-2}) \ldots P(S_1) \tag{6.4}$$

The independence assumption is used to simplify constraint (6.1) into a product involving each success event:

$$P(S_T) = \prod_t^T P(S_t|S_{t-1}) \geq 1 - \epsilon \tag{6.5}$$

Before discussing evaluation of the simplified chance constraint in Eq. (6.5) and integration with FOX, I discuss its convexity in the next section.

6.2 Single Variate Gaussians in 2D Workspaces

Convexity of a constraint is an important consideration for the efficiency with which the resulting trajectory optimization can be solved. If the feasible region of the constraint is convex, the constraint can be used in a convex program. Convex programs are advantageous in terms of the solvers' efficiency and guarantees on the solutions (local optima are also global optima). Certain instances can be solved in polynomial time [18]. A major advantage of using CDF-based approximations to the chance constraint is that many forms of the joint chance constraint are convex. This means that not only can the chance constraint be rapidly evaluated using an off-the-shelf numerical library but it is also amenable to convex programming.

In the simple case where the agent is bounded on one side by an obstacle, the chance constraint at a single time step can be written in terms of a linear constraint on a single random variable X_t':

$$P\left(\mathbf{x}_t \in C_t\right) = P\left(a_t X_t' \leq b_t\right) \tag{6.6}$$

where a_t and b_t relate to the obstacle's surface geometry and are determined by FOX's constraint surfaces. The constraint can be rewritten in terms of the cumulative distribution of w, F:

$$P\left(X_t' \leq b_t/a_t\right) = F\left(X_t' \leq b_t/a_t\right) \geq 1 - \epsilon \tag{6.7}$$

Plugging (6.7) into (6.5), the chance constraint can be written as the product of cumulative distribution functions:

$$\prod_t^T F\left(X_t' \leq b_t/a_t\right) \geq 1 - \epsilon \tag{6.8}$$

This constraint is expressible in the form $g(x) \geq 0$, which means $g(x)$ must be a concave function in order for the feasible set to be convex [20]. The following theorem is stated regarding the concavity of (6.8):

Theorem 6.2.1. *For $\epsilon < 0.5$ and when $\mathbf{w}$ are i.i.d. Gaussian, the feasible set of the constraint in* (6.8) *is convex.*

The theorem follows immediately from a proof in [7]. Specifically, [7] investigates probabilistic constraints of the form $F(a) = P(X \leq a)$. It is shown that the feasible set of this constraint is convex given certain restrictions on the density f and ϵ. For Gaussian distributions, $\prod_t F_t(a_t)$ is concave when $\epsilon < 0.5$. As noted in [80], most path planning missions will desire less than a 50% chance of failure, so the restriction is not greatly limiting.

6.2.1 Two obstacle case

In many situations, an agent may be traveling between two obstacles. The one-sided chance constraint, (6.8), must be modified to model the second obstacle:

$$P\left(o_l \leq X' \leq o_u\right) \geq 1-\epsilon \tag{6.9}$$

where o_l is the upper boundary of the lower obstacle and o_u is the lower boundary of the upper obstacle. For one waypoint, Eq. (6.9), can be modeled in terms of F:

$$P\left(o_l \leq X' \leq o_u\right) = F\left(\frac{o_u-\bar{x}'}{\sigma}\right) - F\left(\frac{o_l-\bar{x}'}{\sigma}\right) \geq 1-\epsilon \tag{6.10}$$

In the case of Gaussian uncertainty, it can be shown that the feasible set of (6.10) is convex [73]. Under the independence assumptions stated above, the joint chance constraint can be written as the product of the individual chance constraints:

$$\prod_t P\left(o_{t,l} \leq X_t' \leq o_{t,u}\right) = \\ \prod_t F\left(\frac{o_{t,u}-\bar{x}_t'}{\sigma}\right) - F\left(\frac{o_{t,l}-\bar{x}_t'}{\sigma}\right) \geq 1-\epsilon \tag{6.11}$$

In the case of Gaussian uncertainty, it is straightforward to show that the feasible set of (6.10) is convex. Two observations are made of the Gaussian cumulative distribution and its second derivative with respect to μ:

Observation 6.2.2. In the case of Gaussian uncertainty, the probability of success at a single time step, $P\left(X' < a\right)$, can be written in terms of the error function as:

$$F\left(a;\mu_t,\sigma\right) = 0.5\left(1+\operatorname{erf}\left(\frac{a-\mu}{\sqrt{2}\sigma}\right)\right) \tag{6.12}$$

Observation 6.2.3. The second derivative of (6.12) with respect to μ is:

$$\frac{d^2F}{d\mu^2} = -\frac{1}{\sqrt{2\pi}\sigma}\left(\frac{a-\mu}{\sigma^2}\right)\exp\left(\frac{-\left(a-\mu\right)^2}{2\sigma^2}\right) \tag{6.13}$$

Using Obs. 6.2.3, the second derivative of (6.10) is:

$$\frac{d^2F}{d\mu^2} = -\frac{1}{\sqrt{2\pi}\sigma}\left(\frac{o_u-\mu}{\sigma^2}\right)\exp\left(\frac{-\left(o_u-\mu\right)^2}{2\sigma^2}\right) \\ -\left(-\frac{1}{\sqrt{2\pi}\sigma}\left(\frac{o_l-\mu}{\sigma^2}\right)\exp\left(\frac{-\left(o_l-\mu\right)^2}{2\sigma^2}\right)\right) \tag{6.14}$$

For $\epsilon < 0.5$, we have $\mu < o_u$ and $\mu > o_{t_l}$, implying that (6.14) is negative. Hence, the feasible region of (6.10) is convex.

Given that $\mathbf{w}$ is composed of *i.i.d.* random variables, the joint chance constraint can be written as the product of the individual chance constraints:

$$\prod_t P\left(o_{t,l} \leq X'_t \leq o_{t,u}\right) = \prod_t F\left(\frac{o_{t,u}-\mu_t}{\sigma}\right) - F\left(\frac{o_{t,l}-\mu_t}{\sigma}\right) \geq 1-\epsilon \tag{6.15}$$

The joint chance constraint's feasible region is also convex. The following theorem is stated:

Theorem 6.2.4. *For* $\epsilon < 0.5$ *and when* W *are i.i.d. Gaussian, the feasible set of the constraint in* (6.15) *is convex.*

Proof. The proof presented in [7] is used as guidance. It is shown that for any two-sided chance constraint, Eq. (6.15), it is possible to construct a series of one-sided chance constraints with matching cumulative distributions and first derivatives. Using arguments from [7], the Hessian of the one-sided chance constraint is shown to be negative semidefinite. This implies that the Hessian of the two-sided chance constraint is negative semidefinite and that the feasible set of (6.15) is convex.

First, note that the negative Hessian of (6.15) is:

$$H(\boldsymbol{\mu}) = -\begin{bmatrix} F''(\mu_0)\prod_{t=1}F(\mu_t) & \dots & F'(\mu_0)F'(\mu_n)\prod_{t\neq 0,n}F(\mu_t) \\ \vdots & \ddots & \vdots \\ F'(\mu_0)F'(\mu_n)\prod_{t\neq 0,n}F(\mu_t) & \dots & F''(\mu_n)\prod_{t\neq n}F(\mu_t) \end{bmatrix} \tag{6.16}$$

where $F'(\mu_t)$ $F''(\mu_t)$ are the first and second derivatives of the individual chance constraint taken with respect to μ_t. For convexity, this must be positive semidefinite;

it must have positive eigenvalues. As shown in [7], this can be rewritten,

$$H(\boldsymbol{\mu}) = -F(\boldsymbol{\mu}) \begin{bmatrix} \frac{F''(\mu_0)}{F(\mu_0)} & \cdots & \frac{F'(\mu_0)F'(\mu_n)}{F(\mu_0)F(\mu_n)} \\ \vdots & \ddots & \vdots \\ \frac{F'(\mu_0)F'(\mu_n)}{F(\mu_0)F(\mu_n)} & \cdots & \frac{F''(\mu_n)}{F(\mu_n)} \end{bmatrix} \tag{6.17}$$

where $F(\boldsymbol{\mu})$ is the original constraint, Eq. (6.11). We show it is positive definite by comparison with a one-sided chance constraint.

Given a two-sided individual chance constraint, we can always create an equivalent one-sided chance constraint that matches the two-sided chance constraint and its first derivative. Precisely, for a two-sided chance constraint, F_2 , we set $F_2 = F(a) - F(b) = F_1(z)$ where $a = (o_{t,u} - \mu_t)/\sigma$ and $b = (o_{t,l} - \mu_t)/\sigma$ (i.e. Eq. (6.10)) and F_1 is the one-sided cumulative distribution function. Solving for z:

$$z = F^{-1}(F(a) - F(b)) \tag{6.18}$$

In the case of Gaussian uncertainty, we have the freedom to choose σ_1 and can also match the first derivatives:

$$F_1' = -\frac{1}{\sqrt{2\pi}\sigma_1} \exp -z^2/2 = \frac{dF(a)}{d\mu} - \frac{dF(b)}{d\mu} = F_2' \tag{6.19}$$

This can be solved for σ_1 such that the equality holds. The second derivative is the remaining component of the constraints, which only appears on the Hessian's diagonal. Given the above choices, $F_1'' > F_2''$. The second derivative of the two-sided chance constraint is given by (6.14). The second derivative of F_1 is:

$$F_1'' = -\frac{z}{\sqrt{2\pi}\sigma_1^2} \exp\left(-z^2/2\right) \tag{6.20}$$

Substituting for σ_1 from Eq. (6.19), and canceling a $\exp(-z^2/2)$ in the numerator:

$$F_1'' = -\frac{z\left(\exp(-a^2/2) - \exp(-b^2/2)\right)^2}{\sqrt{2\pi}\sigma^2 \exp(-z^2/2)} \tag{6.21}$$

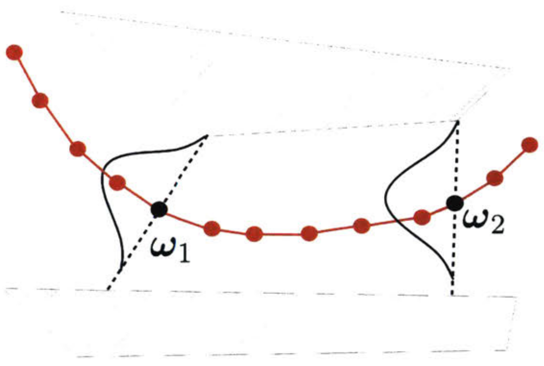


Figure 6-1: Illustration of FOX's chance constraint computation at waypoints along a trajectory. Red dots are knots in the trajectory optimization; black dots are waypoints. The dashed lines represent FOX's constraint segments.

From (6.21) it is possible to factor terms $c_1 = (\exp(-a^2/2) - \exp(-b^2/2))\,/\,(\exp(-z^2/2))$ and $c_2 = z(\exp(-a^2/2) - \exp(-b^2/2))$. First, note that because of (6.18) and the fact that $\epsilon \leq 0.5$, it must be the case that $z \leq a$ and $z \geq b$ where $b \leq 0$. This implies $\exp(-a^2/2) - \exp(-b^2/2) \leq \exp(-z^2/2)$ or $c_1 \leq 1$. Comparing c_2 to the two-sided second derivative, Eq. (6.14), we must show that $c_2 = z\exp(-a^2/2) - z\exp(-b^2/2) \leq a\exp(-a^2/2) - b\exp(-b^2/2)$. Given the constraints on z, a, and b, this is the case. This shows that $F_1'' \geq F_2''$ given the constraints.

From the arguments in [7], the Hessian of $-F_1$ is positive semidefinite. Because $F_1 = F_2$ and $F_1' = F_2'$, the Hessians are identical aside from the main diagonal. Furthermore, we have shown that $-F_2'' > -F_1''$; this implies the main diagonal elements of H_2 are greater than H_1. This implies that its eigenvalues are greater than those of H_1 and H_2 is also positive semidefinite. $\square$

This proves that the two-obstacle chance constraint can be formulated to have a convex feasible region. The mean of the state μ_T must simply be constrained to be closer to the obstacle above or below. The fact that the feasible regions is convex means that the constraint can be passed to convex optimization routines.

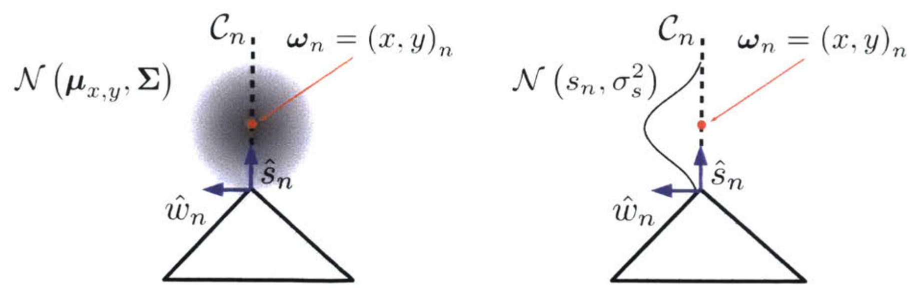

$\mathcal{C}_n$
$\boldsymbol{\omega}_n = (x, y)_n$
$\mathcal{N}(\boldsymbol{\mu}_{x,y}, \boldsymbol{\Sigma})$
$\hat{s}_n$
$\hat{w}_n$
$\mathcal{C}_n$
$\boldsymbol{\omega}_n = (x, y)_n$
$\mathcal{N}(s_n, \sigma_s^2)$
$\hat{s}_n$
$\hat{w}_n$

Figure 6-2: Illustration of variables, projection, and bounds relevant to the risk computation at each waypoint. The left image represents the bivariate state distribution centered at the waypoint $\boldsymbol{\omega}_n$. The right image depicts the projection of the distribution onto the constraint segment.

6.3 Chance Constraints with FOX

FOX's constraint surfaces form not just waypoints for the motion planner but also geometric constraints with which to evaluate a CDF-based chance constraint. To compute the CDF, it's necessary to have bounds with which to integrate as well as a fully parameterized distribution. These are provided by FOX's constraint surfaces. Specifically, in 2D, the waypoint position states $\boldsymbol{\omega} = (x, y)$ are projected onto a constraint segment with basis $(\hat{s}, \hat{w})$. Coordinate $\hat{s}$ runs parallel the constraint segment and $\hat{w}$ is orthogonal to the segment. As mentioned in Chapter 4, the projection of $(x, y)_n$ onto $\mathcal{L}$ is linear:

$$s_n = (\boldsymbol{\omega}_n - \vec{o}_n) \cdot \hat{s}_n \tag{6.22}$$

$$w_n = (\boldsymbol{\omega}_n - \vec{o}_n) \cdot \hat{w}_n \tag{6.23}$$

The waypoint constraint from Chapter 4 requires the position state lie on the constraint segment:

$$w_n = 0 \;\; \forall \;\; n \in N \tag{6.24}$$

To simplify the chance constraint integration, the distribution describing the agent's position state is projected onto the constraint segment and integrated along the $\hat{s}$-coordinate. The mean of the projected distribution is s_n from Eq. (6.22); it is the distance of the waypoint to the obstacle surface. This idea of computing the chance

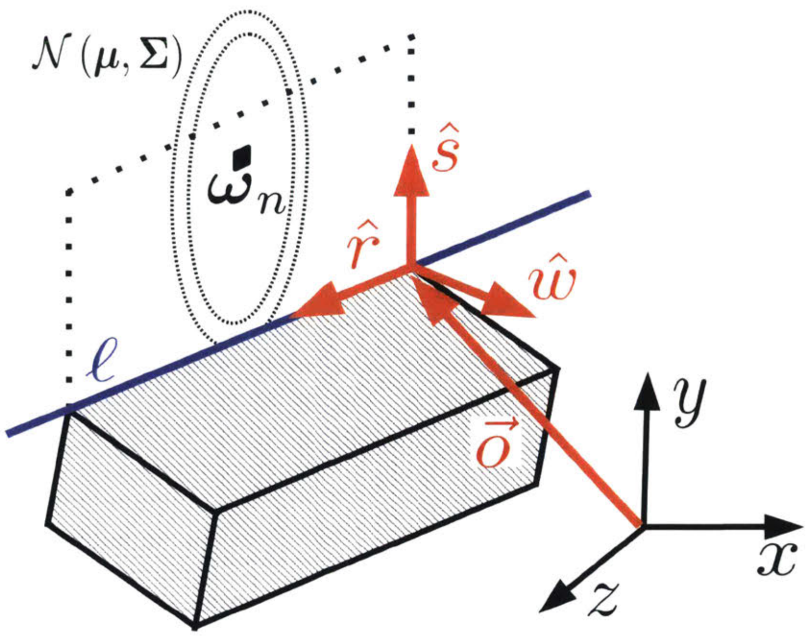


Figure 6-3: Illustration of the projection of the state distribution onto the constraint plane for a 3D workspace. The dotted ellipses indicate the projection of the multivariate Gaussian onto the plane. The distribution is then integrated from the line ℓ.

constraint by integrating with respect to the agent's distance from an obstacle is used in a similar manner in [15]. The integration required to evaluate the chance constraint at a single waypoint, the probability of success $P(S)$, is:

$$P(S) = \frac{1}{\sqrt{2\pi}\sigma_s} \int_0^{\infty} \exp\left(\frac{-(s - s_n)^2}{2\sigma_s^2}\right) ds \tag{6.25}$$

This is used with Eq. (6.5) to evaluate the joint chance constraint. The variables relevant to chance constraint computation as well as the projection are depicted in Fig. 6-2.

6.3.1 Multivariate Gaussians in 3D Workspaces

Under multivariate Gaussian uncertainty, the chance constraint becomes:

$$\bigwedge_{n \in T} P(\mathbf{x}_n \in \mathcal{L}_{n,convex}) = \prod_n \Phi_{\Sigma,\boldsymbol{\ell}} \geq 1 - \epsilon \tag{6.26}$$

where $\Phi_{\Sigma,\boldsymbol{\ell}}$ is the cumulative distribution function of the multivariate Gaussian distribution with correlation matrix Σ integrated with respect to bounds $\boldsymbol{\ell}$. While this integral is difficult to evaluate, very efficient integration routines have been implemented such as those described in [36]. In terms of (r, s) variables, the constraint may be written using the bivariate normal distribution as [116]:

$$\Phi_{\Sigma,\boldsymbol{\ell}} = \int_{-\infty}^{\infty} \int_{0}^{\infty} \frac{1}{2\pi\sigma_s\sigma_r\sqrt{1-\rho^2}} \exp\left(-\frac{z}{2\left(1-\rho^2\right)}\right) dsdr \tag{6.27}$$

$$z = \frac{\left(s-\mu_s\right)^2}{\sigma_s^2} - \frac{2\rho\left(s-\mu_s\right)\left(r-\mu_r\right)}{\sigma_s\sigma_r} + \frac{\left(r-\mu_r\right)^2}{\sigma_r^2} \tag{6.28}$$

where ρ is the correlation coefficient. In this expression μ_s and μ_r are the decision variables determined by the optimizer (i.e. the waypoints) after being transformed from the global coordinates $\bar{\mathbf{x}}_n = (x, y, z)_n$ using transformations (4.12) - (4.14). The bounds of integration are determined by $\mathcal{L}$; the lower bound of the inner integral is at $s = 0$, i.e. the line $\vec{\ell_0}$ as shown in Fig. 6-3. This bound is associated with the generating edge from FOX. Other bounds such as $\vec{\ell_1}$ in Fig. 6-3 are generated from other obstacles with which the constraint surface may collide. Intuitively, the closer a waypoint to any of the bounds in the set $\boldsymbol{\ell}$, the riskier the waypoint.

6.4 Test Cases & Benchmarks

The algorithm was benchmarked against two other approaches from literature. First, a benchmark was performed against the chance-constrained RRT approach discussed in [74]. This algorithm extends the popular RRT planning algorithm to the probabilistic case. In brief, points are sampled from the state space and a set of nearest neighbors computed, which currently lie on the search tree. Paths are extended from the neighbors in an attempt to connect with the new sample. The probabilistic version assumes not only that paths are collision free but respect a chance constraint. The reader is referred to the innovative work in [74] for full details.

A second benchmark was performed against a disjunctive linear programming approach similar to that presented in [12]. This work considered convex, two dimensional

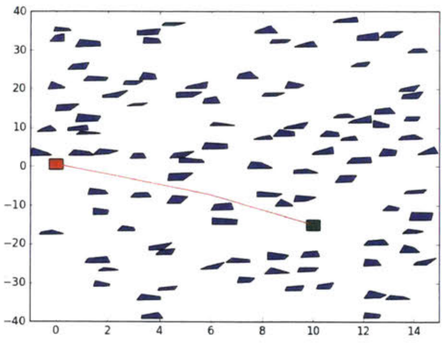


Figure 6-4: Example 100 trapezoidal obstacle test case. Dynamics are a simple 2D double integrator.

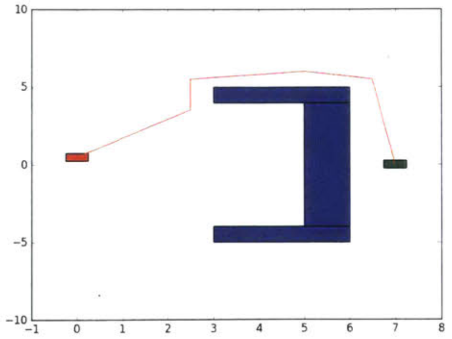


Figure 6-5: Example of the algorithm avoiding a local minimum "bugtrap" in 2D.

obstacles. In this setting, $\mathcal{W}_{free}$ can be described by a conjunction of disjunctions:

$$\mathbf{x}_t \in \mathcal{W}_{free} \;\; \forall \;\; t \iff \bigwedge_t \bigvee_l \mathbf{a}_{jl}\mathbf{x}_t \geq b_{jl} \tag{6.29}$$

where l represents the l^{th} linear constraint of the j^{th} obstacle and $\mathbf{a}$ and b are constants relating to surface geometry. Because of the disjunction, an integer decision variable $z_{j,t,l}$ is required that encodes whether the j^{th} obstacle has constraint l active at time step t:

$$\mathbf{a}_{jl}\mathbf{x}_t - \mathbf{b}_{jl} + Mz_{j,t,l} \leq 0 \tag{6.30}$$

where M is a big positive constant. Constraint (6.30) necessitates the use of a MILP solver; our benchmarks relied on [41].

A number of test cases were run. Two dimensional test cases were performed on toy problems, illustrating the algorithm's performance on simple convex obstacles and its ability to avoid local minima. A more realistic three-dimensional test case was run simulating a Slocum Glider AUV navigating the Hawaiian Islands [115]. For this test case, environmental GIS data of the Hawaiian Islands was taken from [43]. The data was manually downsampled so that key environmental features could be captured without generating too many unnecessary constraints. A linearized AUV model with double integrator dynamics (Fig. 6-7) was used for benchmarking.

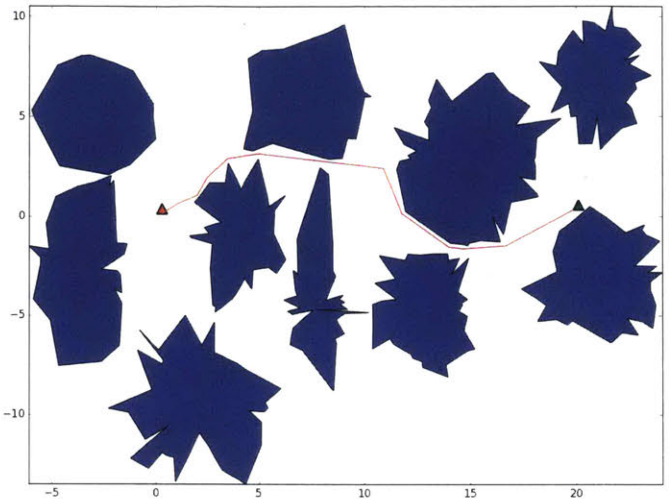


Figure 6-6: FOX makes no assumptions regarding obstacle convexity, which means it can handle more complex obstacles than other approaches.

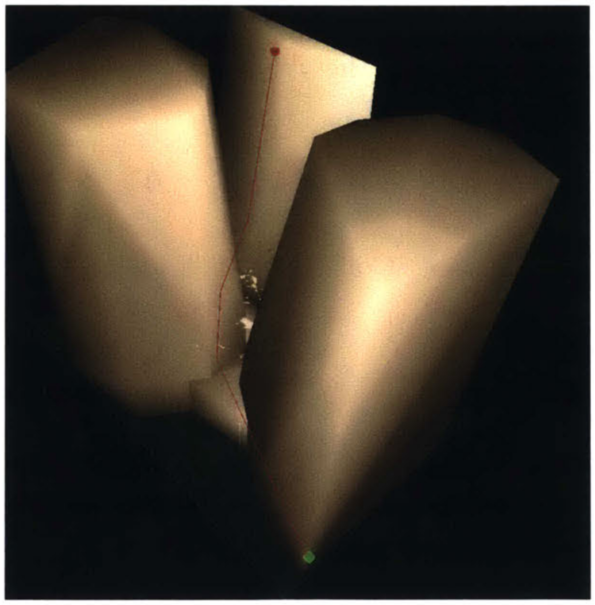


Figure 6-7: A linearized Slocum Glider navigating the Hawaiian islands. The red line indicates a trajectory for AUV. The red cube indicates the starting point; the green cube indicates the goal.

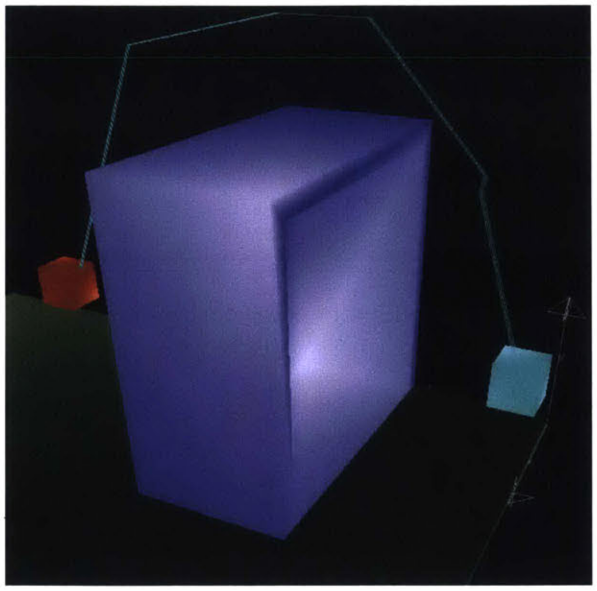


Figure 6-8: Example path avoiding a 3D local minimum, the "3D Box Test." The agent travels from the cyan cuboid to the red cuboid and must avoid paths that get stuck inside the blue box obstacle.

Tests were performed on a desktop computer with an Intel i7-4790 CPU and 16 GB of RAM. All implementations were done in C++ and relied heavily on the Computational Geometry Algorithms Library (CGAL), specifically [3]. SNOPT was used as an off-the-shelf optimization subroutine [37].

6.5 Results

Example trajectories produced by my algorithm are shown in Figs. 6-4 - 6-8. Benchmarking plots are provided in Figures 6-9 - 6-12. Plots 6-10 - 6-12 denote the time taken to find an initial feasible solution for both FFHS and CC-RRT with dashed vertical lines. Gradually, the cost of the incumbent best path improves as the algorithms continue the search.

As shown in Figure 6-9, both CC-RRT and hybrid search, FFHS, outperformed the disjunctive linear program (DLP) considerably in terms of speed. One reason for this is that, as presented in [12], the DLP lacks a heuristic which dictates which obstacle constraints may be relevant. This means that the DLP must solve a MILP which accounts for all obstacle constraints in the environment. Large obstacle fields

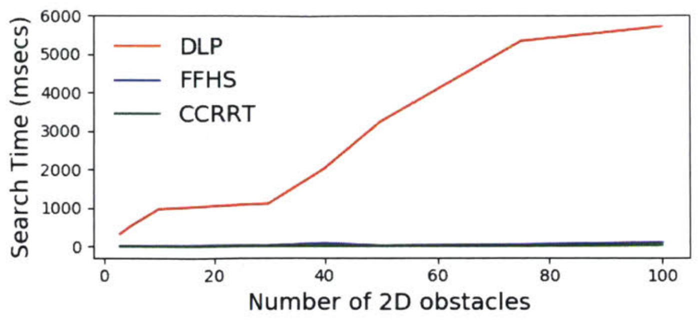


Figure 6-9: Comparison of the time taken to find initial feasible solutions for environments with various numbers of 2D convex obstacles.

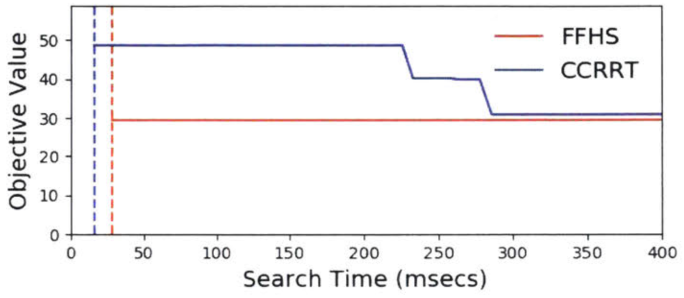


Figure 6-10: Comparison of the performance of FFHS vs. CC-RRT on 2D obstacle test cases with 100 obstacles. Vertical dashed lines indicate the time to find the first feasible solution.

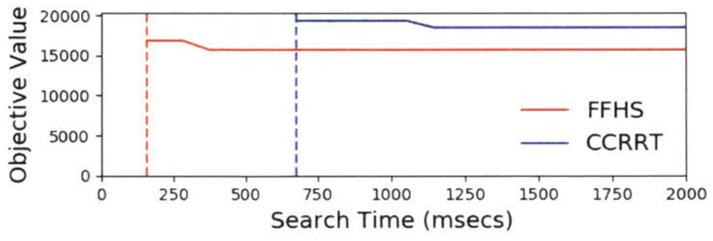


Figure 6-11: Comparison of the performance on a 3D test case where an AUV with linearized dynamics navigates environments generated from Hawaiian Island bathymetry data.

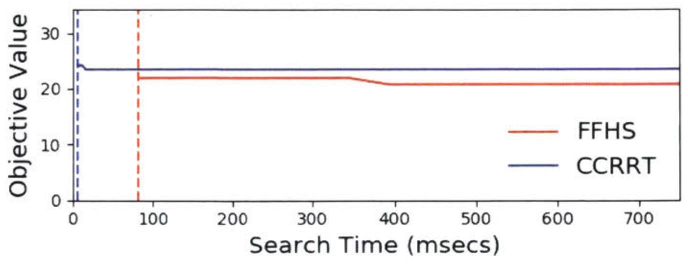


Figure 6-12: Comparison of the performance of FFHS vs. CC-RRT on the "3D Box Test" where the agent must navigate around a local minimum.

will generate enormous mixed integer linear programs even when only a small number of the constraints are relevant.

Fig. 6-9 shows that CC-RRT and FFHS are close in performance with CC-RRT having a slight advantage in time taken to find an initial feasible solution. Similar results are seen in Figs. 6-10 and 6-12 where CC-RRT is able to generate an initial feasible solution faster. On the other hand, FFHS performs well when comparing the quality of solutions. Typically, FFHS can find a solution almost as fast as CC-RRT but the solution is typically much better. The worst performance of FFHS vs. CC-RRT is on the "3D Box Test." This is likely because FFHS must take time to get out of the local minimum while CC-RRT can quickly connect sampled points outside the box. FFHS does very well on the Hawaiian Island test case and even finds an initial feasible solution faster than CC-RRT. This is likely because the environment is very large with a small number of obstacle surfaces that must be avoided. FFHS shoots paths directly towards the goal and is able to generate the small number of active obstacle constraints. CC-RRT spends time expanding long paths into regions that are not helpful in finding the goal.

Overall, the results show that both CC-RRT and FFHS are efficient at rapidly generating feasible, risk-bound trajectories. The results show that there is value in the hybrid search and using a trajectory optimizer to improve paths. The hybrid search often leads to considerably better paths in similar amount of time as the RRT-based approach.

Chapter 7

Risk for Agents with Geometry

One of the drawbacks of the previous chapter is that FOX is only applicable to point robots. This is unrealistic for many real world systems where dealing with the agent's geometry is important to generate a probabilistically feasible motion plan. This chapter discusses an extension to FOX that enables the approach to compute the probability of collision for agents with non-trivial geometry. It is a challenging problem because computing the probability of collision for an agent with non-trivial geometry requires potentially complex limits of integration. The key insight used in this chapter is that a difficult integration can be transformed into a manageable computational geometry computation. The computational geometry computation can be formulated in a way that makes it amenable to the trajectory optimization approaches described in previous chapters.

7.1 Modeling the Risk of Agents with Non-Trivial Geometry

Approximating the probabilistic constraint is difficult because of the need to integrate probability distributions with respect to complex limits of integration. This is especially true in the case of agents with non-trivial geometry (i.e. agents which are modeled as polytopes rather than points). A simple example is shown in Fig. 7-1.

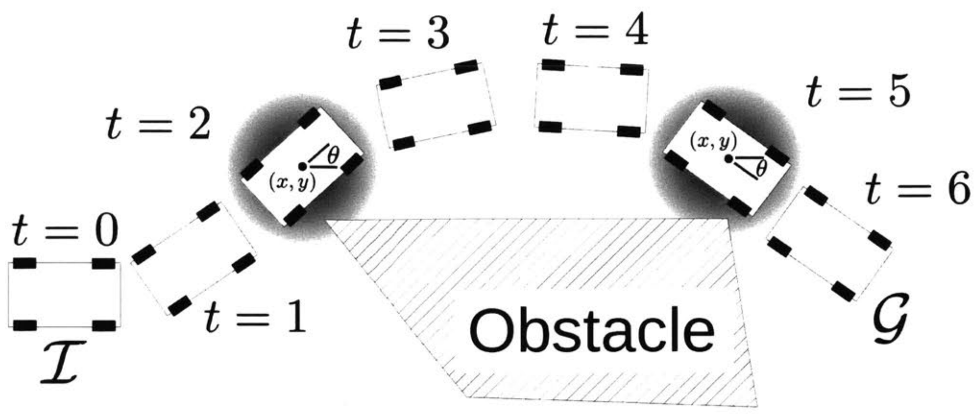


Figure 7-1: Computing the risk of even a simple rectangular shape is challenging. A possible distribution of the agent's state at $t = 2$ and $t = 5$ is shown in gray. Not only is there uncertainty in position states (x, y) but also orientation θ. An integration to compute the collision risk must be performed over all configurations that lead to a collision.

Computing the collision probability involves integrating a multivariate density with respect to limits ℓ, which are a function of the agent and obstacle's geometry. Computing these limits is challenging because it is non-trivial to compute whether a given agent configuration is in collision with an obstacle. For the simple 2D Dubins car, this computation is a function of the three configuration states x, y, θ, and, assuming the car is modeled as a rectangle, the length l and width w of the car. It is also a function of the obstacle surface. Once these functions are determined analytically, the integral must typically be broken into parts and evaluated numerically. While [36] presents many efficient techniques for numerically approximating the multivariate normal cumulative distribution function, complex limits of integration remain challenging to solve.

Consider the rectangular polygon that models a simple car as shown in Fig. 7-1. The configuration states of the rectangle are $\mathcal{C} = (x, y, \theta)$, which fully specify the position of the rectangle in the workspace. Additionally, assume that the probability distribution describing these states is uniform, $f(\mathcal{C}) \sim \mathcal{U}$, in some box (x, y) and $0 \le \theta < 2\pi$:

$$f_{\mathcal{U}}(x, y, \theta) \begin{cases} \frac{1}{2\pi ab}, & \text{for } 0 \le x < a, 0 \le y < b, 0 \le \theta < 2\pi \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

In this case,

$$P(S_k) = \frac{1}{2\pi ab} \int_{\ell} \mathbb{1}_{q \in \mathcal{C}_{free}} dq \tag{7.2}$$

where $\mathcal{C}_{free}$ is the free region of the agent's configuration space. The integral in Eq. (7.2) is equivalent to determining the volume of the free region of the agent's configuration space over the support of the distribution $f_{\mathcal{U}}$. In general, this is a very difficult region to determine, not mention compute its volume [69].

Approach

Rather than performing the CDF-based computations outlined in Chapter 6, I propose a numerical approach. The approach relies on combining a collision checking routine with Monte Carlo sampling to compute collision risk. First, rather than computing whether an agent configuration is in collision analytically, a collision checker is used. As mentioned in Chapter 4, collision checking algorithms are a mature field for which many efficient options exist. FOX's heuristic-based search already assumes the availability of an efficient collision checking routine.

Given the ability to test whether an agent configuration is in collision or not, it is necessary to transform this information in a way that can be used by FOX's trajectory optimization subroutine. This is non-trivial because collision checks are step functions; their derivative is a Dirac delta function and is non-zero only at the point of collision. Because trajectory optimizers rely on information from the Jacobian matrix in order to satisfy the problem's constraints, simply using a step function in an optimizer does not work well. Therefore, rather than using a collision check directly as a constraint, this chapter uses the intersecting region, $\mathcal{W}_{int}$, between the agent and obstacles as a proxy for whether an agent configuration is in collision.

Finally, it is necessary to model the probability of collision. This is done numerically using a Monte Carlo approach. A large number of samples of the agent's state

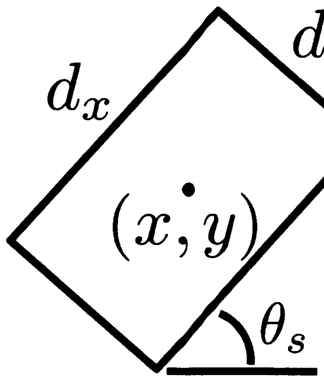


Figure 7-2: Geometry and state variables of rectangle used to model the Dubins car.

distribution are drawn and the intersecting region $\mathcal{W}_{int}$ computed for each sample. To model the failure or success (i.e. collision or lack of collision) of each sample as an indicator variable, the intersecting region $\mathcal{W}_{int}$ is input into a sigmoid. The number of samples that fail over the total number of samples provides an estimate of the probability of failure.

7.2 Dynamical Models

To illustrate the algorithm, I use two dynamical test cases, one 2D and the other 3D. A rectangle is used to model a Dubins car in two dimensions and a cuboid is used to model a Slocum glider in three dimensions.

7.2.1 Dubins Car in 2D

In two dimensions, I use the well-known Dubins car model. The Dubins car is often used to illustrate control and trajectory planning algorithms on the simplest possible nonlinear dynamical model [69]. The equations of motion are:

$$\dot{x} = cos\,(\theta) \tag{7.3}$$

$$\dot{y} = sin\,(\theta) \tag{7.4}$$

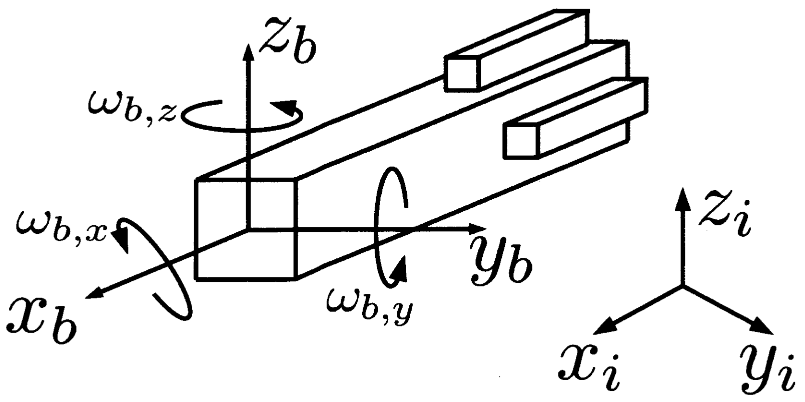


Figure 7-3: Overview of the polyhedron finned Slocum glider model used in this thesis. Coordinates (x_i, y_i, z_i) represent inertial coordinates, (x_b, y_b, z_b) represent body coordinates, and angular velocities $(\omega_{b,x}, \omega_{b,y}, \omega_{b,z})$ represent rotations around the respective body coordinates.

$$\dot{\theta} = u_\theta \tag{7.5}$$

where (x, y) are 2D position states and θ is the steering angle. The change in steering angle is controlled via u_θ. While the Dubins car is often modeled with speed as a second control input, I assume this is 1 for simplicity.

Typically, Dubins cars are modeled as points. To illustrate FOX's ability to deal with geometry, I model the car's geometry as a rectangle with length and width d_x and d_y. The state's position (x, y) is placed at the car's center of area. The geometry and state variables are depicted in Fig. 7-2.

7.2.2 Slocum Glider in 3D

A Slocum glider is used to illustrate the 3D test case. The Slocum glider is an underwater autonomous vehicle (AUV) that requires very little battery energy due to the presence of a buoyancy engine. The buoyancy engine uses the ocean's temperature gradient (thermocline) to generate an oscillatory movement. The oscillatory movement allows the AUV to glide forward. In effect, the AUV uses energy stored in the ocean for forward propulsive movement, giving the glider a very high efficiency. Ultra-high efficiency means the glider is capable of long duration missions [29].

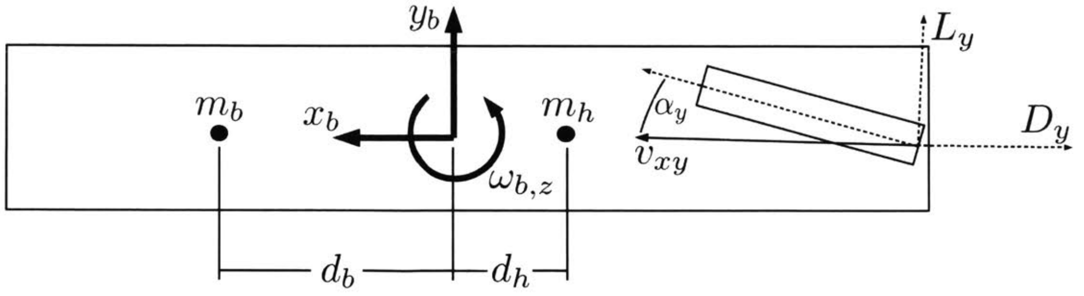


Figure 7-4: View of glider forces and geometry in the xy-plane. The xz-plane is similar.

A model of the glider's dynamics must take into account the changes in linear and rotational momentum. This thesis closely follows the glider model in [70]. This work derives the equations of motion by analyzing the glider's kinetic energy and momentum. The chief difference between the model used in this thesis and in [70] are the control inputs. The model in [70] assumes the glider has access to two control inputs: a stationary point mass whose mass value can be varied and a movable point mass. The stationary point mass models the buoyancy engine; the glider's overall buoyancy is altered by changing its mass. Moving the movable point mass affects the center of buoyancy and consequently the glider's pitch. To simplify the model in this thesis, I assume the glider has a variable mass but uses adjustable fins to change pitch and yaw rather than a movable mass. This removes the movable point mass's state from the equations of motion and allows for a slightly simpler model. An overview of the glider model's geometry and coordinate system is shown in Fig. 7-3.

While [70] focuses on control of a 2D test case, this thesis uses a full 3D dynamical model as a test case. It relies on thirteen states and three control inputs. The states are:

$$[x_i, y_i, z_i, v_{x,b}, v_{y,b}, v_{z,b}, \psi, \theta, \phi, \omega_x, \omega_y, \omega_z, m_b] \tag{7.6}$$

States (x_i, y_i, z_i) are the glider's position states in the inertial reference frame, while states $(v_{x,b}, v_{y,b}, v_{z,b})$ are velocity states in the glider's body reference frame. Angles (ψ, θ, ϕ) are Euler angles that transform the inertial frame to the body frame by a rotation ϕ about the original z axis, a rotation θ about the new y axis, and finally a rotation ψ about the new x axis as described in [106]. States $(\omega_x, \omega_y, \omega_z)$ are the rates of rotation around the x, y, and z axes in the body frame. m_b is the mass of the buoyancy engine. I assume three control inputs: u_{pitch}, u_{yaw}, and $u_{m_{buoyancy}}$. Inputs u_{pitch} and u_{yaw} model input fin angles; $u_{m_{buoyancy}}$ is the change in mass of the buoyancy engine.

Because description of the full dynamical model is long and it is not the main focus of this chapter, the model and its derivation are described in Appendix A.

7.2.3 Assumptions and Notations

This chapter makes the same assumption as Chapter 4 in that obstacles and agents are represented via polytopes, which are stored using the data structures described in Section 4.6.1. The term *configuration states* is used to mean any state variable that directly changes the position of the agent in the workspace. For the Dubins car example, this refers to states (x, y, θ) and for the Slocum glider this refers to states $(x, y, z, \psi, \theta, \phi)$. Like previous chapters, $\mathcal{W}$ is used to represent regions of the workspace, i.e. $\mathcal{W} \in \mathbb{R}^2$ for 2D workspaces or $\mathcal{W} \in \mathbb{R}^3$ for 3D workspaces. $\mathcal{W}_a$ is the region associated with the agent and $\mathcal{W}_{obs}$ is the region associated with obstacles. The intersecting area is $\mathcal{W}_{int} = \mathcal{W}_a \cap \mathcal{W}_{obs}$.

7.3 Integrating Collision Checking with SQPs

As described in Section 7.1, the collision risk is modeled by sampling from a probability distribution describing the agent's state and then using a collision checker to determine whether each sample is in collision. This approach is integrated with FOX's polytope model of the environment and agent. A few insights drive this approach:

- Collision checking is fast and reliable.
- Collision checks can handle complex geometry, which helps avoid the need for complicated limits of integration required if the collision risk were computed via integrals.

- Collision checking can be parallelized on graphical processing units (GPUs), which allows the parallel evaluation of large numbers of scenarios.

On the other hand, there are a number of challenges with this approach. To solve the risk-bound motion planning problem, this thesis relies on a trajectory optimization transcribed as a mathematical program. In the case of nonlinear dynamics, the mathematical program is often solved using sequential quadratic programming (SQP) such as implemented in [37]. The SQP method requires evaluation of the Jacobian matrix, which is the gradient of the objective and constraints with respect to the decision variables. Intuitively, the SQP method uses the gradients to help produce a feasible solution. SQP routines often work best if the user supplies analytic expressions for the Jacobian matrix; otherwise, numerical approximations are made. Because of this, raw collision queries are difficult to integrate into SQP methods. A collision query returns a binary variable based on whether the agent is in collision or not. This is a step function of the agent's configuration state. Its gradient is a Dirac delta function, nonzero only at the point of intersection. A constraint better suited for an SQP solver would have a non-zero partial derivative over a wider range of agent states.

The second challenge with using sampling-based methods to approximate the collision risk is that it is not apparent from which distribution samples should be drawn. One major advantage of using additive Gaussian noise for risk-bound planning with linear dynamics is that the state distribution is known at all future time steps. With a nonlinear system (such as both the Dubins car and Slocum glider), the dynamics are nonlinear and the state distribution non-Gaussian. Therefore, it is inaccurate to simply draw samples from a Gaussian distribution. This problem is the focus of the following chapter. The current chapter focuses on integrating collision checks into the trajectory planner.

In order to provide a better collision gradient for the SQP optimizer, I propose a continuous measure of the intersection between the agent and the obstacles. Specifically, I propose using the region of the workspace where the agent and obstacle intersect, $\mathcal{W}_{int}$ where $\mathcal{W}_{int} = \mathcal{W}_a \cap \mathcal{W}_{obs}$. The size of $\mathcal{W}_{int}$ is used as a proxy for

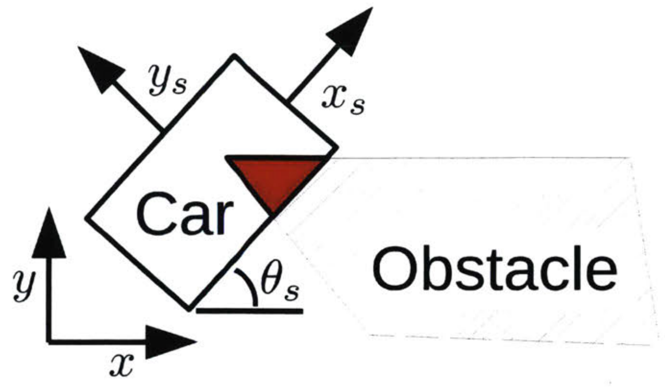


Figure 7-5: One time step of the Dubins car while in collision with an obstacle The Dubins car is modeled as one rectangle out of simplicity. The red triangle represents the intersecting area $\mathcal{W}_{int}$ that must be computed.

constraint violation; in two dimensions this is the intersecting area and in three dimensions it is the intersecting volume. In the case of a collision free path $|\mathcal{W}_{int}| = 0$, and in the case of a colliding path $|\mathcal{W}_{int}| > 0$. An example of this idea is shown on a configuration of the Dubins car in Fig. 7-5.

While I believe this is a novel technique for motion planning under uncertainty, it takes inspiration from prior work. The innovative work in [14] uses particles or samples to approximate the collision probability. Although [14] focuses on the simpler problem of point robots, the strategy is similar: sample from the agent's state distribution and determine whether each sample is in collision. In addition to handling more complex geometry, there is a key difference between this formulation and that presented in [14]. The work in [14] solves the problem as a mixed integer linear program (MILP). Binary decision variables are used to model whether each particle is in collision. While this approximates the chance constraint, it generates a very large number of decision variables and hence a very complicated MILP. In the simple case of a point robot, the number of binary decision variables grows as TS where T is the number of time steps and S the number of samples per time step. Both of these numbers can be large; S can be in the hundreds in order to generate a reasonable approximation. This is in addition to the decision variables required to model other quantities such as the dynamics. On the other hand, for the SQP formulation presented in this chapter, the number of decision variables is not a function of the precision with which the chance constraint is evaluated. This enables a much simpler mathematical program.

This work also uses a similar approach to the influential TrajOpt algorithm [98]. TrajOpt is a trajectory optimization routine that integrates obstacles. TrajOpt is similar in that it also forms a continuous measure of whether an agent is in collision with obstacles. Rather than use the intersecting region, TrajOpt relies on the penetration depth between the agent and obstacles. The penetration depth is a scalar distance defined as the minimum translation distance that would resolve a collision between an agent and obstacles. Unlike the method presented in this chapter, TrajOpt focuses on the deterministic case and convex obstacles.

7.3.1 Computing 2D intersection area

In two dimensions, the intersection of two polygons produces another polygon or, if the intersection occurs at only one point, a point. Computing the intersection is termed *clipping*. In general, computing clippings for general polygons is difficult. In spite of this, many efficient algorithms have been proposed for general polygons such as Vatti's clipping algorithm, presented in [111], and an implementation in [55].

7.3.2 Computing 3D intersection volume

In three dimensions, computation of the intersecting volume is more complicated. Algorithms exist to compute the intersecting volume between two convex polyhedra such as described in [87]. This clipping algorithm decomposes the subject and clipping polyhedra into planar graphs representing vertices and edges. Clipping occurs by searching through the vertices of the subject graph and clipping vertices that lie on one side of a clipping plane.

Limiting the approach to only convex obstacles would considerably limit its application. It is possible to transform arbitrary obstacles into a set of convex obstacles; however, the process of finding a good convexification is a hard problem [30]. A popular technique is to transform environments into collections of cuboids or voxels [53];

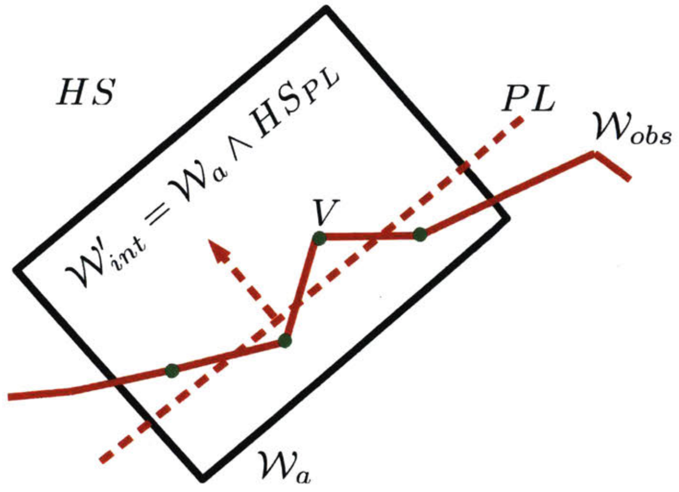


Figure 7-6: Illustration of the method to compute an approximation of the intersecting region $\mathcal{W}_{int}$ for non-convex obstacles. The key is to approximate the obstacle surface that lies inside the agent with a plane, PL. How this plane is constructed depends on the geometry; in the image, a set of vertices $\boldsymbol{V}$ lie inside $\mathcal{W}_a$. The vertices' positions and outward normals are averaged together to generate PL; the outward normal of PL creates the halfspace HS. The intersecting volume is computed as the intersection of this halfspace with $\mathcal{W}_a$.

however, a priori insight is required to properly size the cuboids. I propose a different approach to approximate $\mathcal{W}_{int}$ for convex agents $\mathcal{W}_a$ but possibly non-convex obstacles $\mathcal{W}_{obs}$. The insight is that $\mathcal{W}_{int}$ does not need to be precisely computed. Instead, I compute a proxy for $\mathcal{W}_{int}$, denote this as $\mathcal{W}'_{int}$, that has the following properties. First, $\mathcal{W}'_{int} = 0$ if the sample agent configuration is not in intersection and $\mathcal{W}_{int'} > 0$ if it is. Second, $\mathcal{W}'_{int}$ should increase as the agent and obstacles become more in collision and should decrease as they become less in collision. These criteria suggest the following approach to modeling $\mathcal{W}'_{int}$. Rather than compute the polyhedron $\mathcal{W}_{int}$ via clipping $\mathcal{W}_a$ with possibly non-convex $\mathcal{W}_{obs}$, the facets of $\mathcal{W}_{obs}$ lying inside $\mathcal{W}_a$ are approximated via a plane PL. The intersection of $\mathcal{W}_a$ with PL is convex and its volume can be computed via a standard clipping algorithm.

Specifically, $\mathcal{W}_a$ is the convex polyhedron describing the rigid body of an agent that is in intersection with the not necessarily convex obstacle polyhedron $\mathcal{W}_{obs}$. Let $\boldsymbol{\mathcal{F}}$ be the triangular facets of $\mathcal{W}_{obs}$ which lie inside $\mathcal{W}_a$ at least at one point. There are three cases to consider.

1. At least one vertex V of $\boldsymbol{\mathcal{F}}$ lies inside $\mathcal{W}_a$. If this is the case, all vertices lying inside $\mathcal{W}_a$ are enumerated. Given this set of vertices $\boldsymbol{V}$, the approximating

plane PL is generated by averaging the position and normal vectors of $\boldsymbol{V}$.

2. No vertices of $\boldsymbol{\mathcal{F}}$ lie inside $\mathcal{W}_a$ but edge of at least one facet in $\boldsymbol{\mathcal{F}}$ does. Let the set of all edges of $\boldsymbol{\mathcal{F}}$ that lie in $\mathcal{W}_a$ be $\boldsymbol{E}$. This set of edges is averaged to produce E'; a point on E' defines the point of PL. The the normal vectors of $\boldsymbol{E}$ are averaged to produce the normal vector of PL.

3. Finally, neither vertices nor edges of $\boldsymbol{\mathcal{F}}$ lie in $\mathcal{W}_a$. If this is the case, $\boldsymbol{\mathcal{F}}$ must be of cardinality 1; subsequently, the plane that supports this facet is used as PL.

The normal vector of PL defines a halfspace with which $\mathcal{W}'_{int}$ can be computed via a clipping algorithm. Specifically, $\mathcal{W}'_{int} = \mathcal{W}_a \wedge HS_{PL}$ where HS_{PL} is the halfspace defined by PL.

7.4 Generating the Collision Constraint Gradients

An advantage of modeling the collision constraint using intersecting regions is that the constraint's gradients are relatively straightforward to compute using techniques from vector calculus. Given two intersecting polytopes, it is necessary to compute the change in the intersecting region with respect to the optimization's decision variables. In the case of a collocation trajectory optimization, the decision variables are the agent's states, control inputs, and time step. This is not necessarily the case for other forms of trajectory optimization, as will be discussed in the following chapter. In this chapter, I discuss computation of the collision gradients with respect to a set of *configuration states*; if necessary, other gradients can be derived using the chain rule; this is also discussed in the next chapter.

Only the configuration states affect the agent's position in the workspace. Namely, it is necessary to compute $\partial \mathcal{W}_{int}/\partial x_i$ where x_i is a configuration state. For the Dubins car this means computing:

$$\frac{\partial A_{int}}{\partial x_n}, \frac{\partial A_{int}}{\partial y_n}, \frac{\partial A_{int}}{\partial \theta_n} \;\forall\; n \in N \tag{7.7}$$

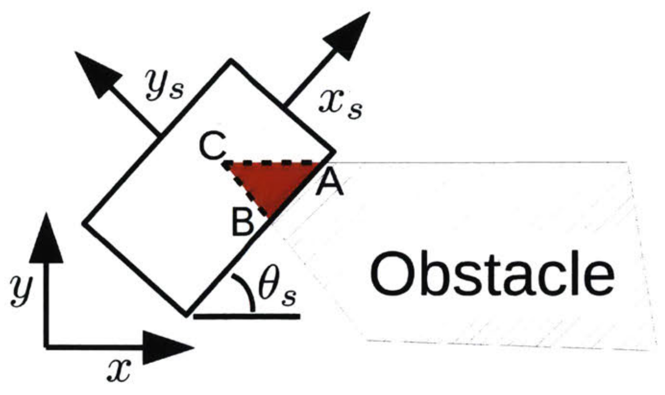


Figure 7-7: The intersecting region $\mathcal{W}_{int}$ is the triangle defined by ABC. The dotted lines are used to compute the partial derivatives of the change in intersection area as a function of the decision variables.

where A_{int} is the intersecting area. For the Slocum glider, it's necessary to compute:

$$\frac{\partial V_{int}}{\partial x_n}, \frac{\partial V_{int}}{\partial y_n}, \frac{\partial V_{int}}{\partial z_n}, \frac{\partial V_{int}}{\partial \psi_n}, \frac{\partial V_{int}}{\partial \theta_n}, \frac{\partial V_{int}}{\partial \phi_n} \ \forall \ n \in N \tag{7.8}$$

where V_{int} is the intersecting volume.

The insight into evaluating these gradients is that they can be computed by integrating the flux Φ over certain interfaces of $\mathcal{W}_{int}$. Flux is an idea from physics that models how much of a substance is flowing through a surface. The total amount of transfer over the surface is computed by integrating a vector field with respect to surface [6]. In this circumstance, flux can be used to compute the amount of incremental area (or volume) entering or leaving $\mathcal{W}_{int}$ due to an incremental change in one of the configuration states. This is exactly the quantity expressed by terms (7.7) and (7.8).

The region flux is computed by integrating some function over select *switching interfaces* of $\mathcal{W}_{int}$. Switching interfaces are edges (or facets in 3D) of $\mathcal{W}_{int}$. On one side of the switching interface is only $\mathcal{W}_a$; on the other side is $\mathcal{W}_{int}$. Intuitively, incrementally changing a configuration state could generate an incremental region flux across each switching interface. This is shown for the Dubins car in collision in Fig. 7-7. For each configuration state (x, y, θ), the flux is computed using a line integral for each switching interface of the intersecting area. For example, the triangle ABC represents the intersecting area. Only segments $\overline{BC}$ and $\overline{AC}$ represent switching interfaces. This is because segment $\overline{AB}$ is entirely embedded in $\mathcal{W}_{obs}$; an infinitesimal change in $\overline{AB}$ does not affect $\mathcal{W}_{int}$. On the other hand, $\overline{BC}$ and $\overline{AC}$ lie at the interfaces between $\mathcal{W}_a$ and $\mathcal{W}_{obs}$. An infinitesimal change in their position will cause an infinitesimal change in $\mathcal{W}_{int}$. The total flux (the total gradient) is the sum of the fluxes over each separating interface i:

$$\frac{\partial \mathcal{W}_{int}}{\partial x} = \sum_s \left. \frac{\partial \mathcal{W}_{int}}{\partial x} \right|_s = \Phi_x = \sum_i \Phi_{x,i} \tag{7.9}$$

where Φ_x means the flux due to the vector field generated by an incremental change in state x. Flux is computed through integration. In $\mathcal{W} \in \mathbb{R}^3$, this is a surface integral:

$$\Phi_i = \int \int_i \mathbf{F}\left(x, y, z\right) \cdot \mathbf{n}\left(x, y, z\right) dS \tag{7.10}$$

where $\mathbf{F}$ is a function that describes the rate of change of points $(x, y, z) \in \mathcal{W}_{int}$ due to changes in the configuration state. The surface normal $\mathbf{n}\left(x, y, z\right)$ is constant across the facets of $\mathcal{W}_{int}$. The integral is carried out over the separating interface i. For $\mathcal{W} \in \mathbb{R}^2$, the flux is computed in a similar manner except that (7.10) is a line integral. The following sections provide some examples computing (7.10) for the test cases.

7.4.1 Generating gradients for $\mathcal{W}^2$

Computing the collision gradients for the Dubins car, modeled as a rectangle, requires integrating Eq. (7.10) as a line integral over edges that are separating interfaces. An illustration of these computations is shown in Fig. 7-8.

The x and y gradients are straightforward to compute:

$$\Phi_x = \left. \frac{dA}{dx} \right|_i = -\int_i \hat{n}_x di = -\hat{n}_x \ell_i \tag{7.11}$$

$$\Phi_y = \left. \frac{dA}{dy} \right|_i = -\int_i \hat{n}_y di = -\hat{n}_y \ell_i \tag{7.12}$$

where subscript i represents the i^{th} switching interface. Variables $\hat{n}_x$ and $\hat{n}_y$ represent the x and y components of outward facing, unit-length normals. Because neither $\hat{n}_x$

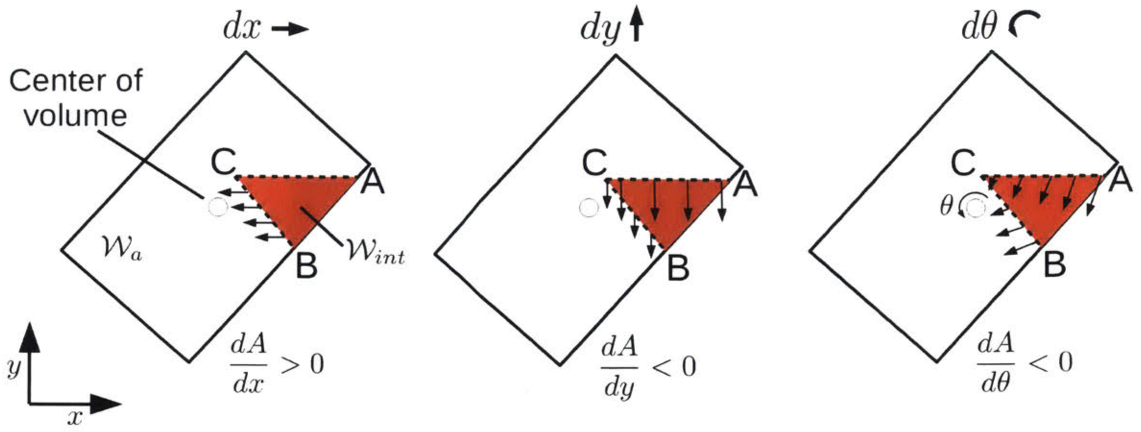


Figure 7-8: Examples of collision gradients for the 2D Dubins car example. Decision variables (x, y) are assumed located at the center of volume; rotations θ are around the center of volume. Arrows indicate the direction that the switching interfaces $\overline{BC}$ and $\overline{AC}$ change due to an incremental change in one of the configuration states. They are opposite the flux. Arrows extending into the white region $(\mathcal{W}_a)$ symbolize flux is heading *into* $\mathcal{W}_{int}$, i.e. the gradient $\partial\mathcal{W}_{int}/\partial$State is increasing. Arrows extending into the red region $(\mathcal{W}_{int} = \mathcal{W}_a \cap \mathcal{W}_{obs})$ indicate that flux is coming *out of* $\mathcal{W}_{int}$, i.e. the gradient $\partial\mathcal{W}_{int}/\partial$State is decreasing.

nor $\hat{n}_y$ change over the edge, the normals may be taken outside the integrals. This results in the integral simplifying to the normal multiplied by the length of the edge, ℓ_i.

The gradient with respect to θ is more difficult to compute because the flux due to rotation varies over the edge. The gradient with respect to a change in θ is:

$$\left.\frac{dA}{d\theta}\right|_i = \int_i \left((c_y - y)\,\hat{n}_y + (c_x - x)\,\hat{n}_x \right) di \tag{7.13}$$

where (c_x, c_y) represent the agent's center of rotation and (x, y) represent x and y points along the line. To evaluate the line integral along the interface i, it is necessary to rewrite x and y in terms of a parameter s that varies along the interface. If the vertices of edge $\overline{AC}$ are (x_0, y_0) and (x_1, y_1), x and y can be written in terms of parameter s as: $x(s) = x_0 + s(x_1 - x_0)$ and $y(s) = x_0 + s(y_1 - y_0)$ where $0 \leq s \leq 1$. This parameterization allows the integral to be rewritten in terms of s:

$$\left.\frac{dA}{d\theta}\right|_i = \int_0^1 (c_y - y_0 - sy_d + c_x - x_0 - sx_d)\sqrt{(x_d)^2 + (y_d)^2}\,ds \tag{7.14}$$

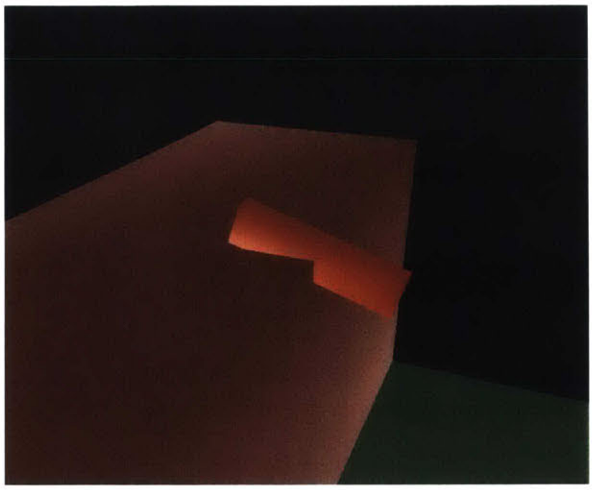


Figure 7-9: A polyhedron representing a simplified model of a Slocum glider (in red) collides with a cuboid obstacle.

where $x_d = x_1 - x_0$ and $y_d = y_1 - y_0$ and the fact was used that $di = \sqrt{(dx/ds)^2 + (dy/ds)^2} ds$. The integral in Eq. (7.14) is straightforward to evaluate.

Given the gradients with respect to each interface, Eqs. (7.11), (7.12), and (7.14), the total gradient is given by their sum over all separating interfaces:

$$\frac{dA}{dx} = \sum_i \left.\frac{dA}{dx}\right|_i \tag{7.15}$$

$$\frac{dA}{dy} = \sum_i \left.\frac{dA}{dy}\right|_i \tag{7.16}$$

$$\frac{dA}{d\theta} = \sum_i \left.\frac{dA}{d\theta}\right|_i \tag{7.17}$$

7.4.2 Generating gradients for $\mathcal{W}^3$

The strategy for determining the gradients for the three dimensional case is similar, although computations are more complicated. The intersecting region $\mathcal{W}_{int}$ is now a polyhedron and the interfaces between the obstacle and the agent are polygons. Surface integrals must be used to compute the flux. In the case of a polyhedron, gradients must be computed with respect to each of the six states in terms (7.8). Similar to the 2D case, the position gradients are simpler to compute than the orientation gradients. Let S represent the surface interface. The x gradient is:

$$\left.\frac{dV}{dx}\right|_i = -\int_S \hat{n}_x dS = -\hat{n}_x A_i \tag{7.18}$$

where the left-hand side indicates that we want the change in volume V with respect to the change in x, $\hat{n}_x$ is the outward facing normal of the i^{th} interface and A_i is the facet's area. As in the 2D case, the flux is constant over the surface so $\hat{n}_x$ can be pulled outside the integral. Similar equations hold for the y and z directions.

For the orientations, the integral is more complicated. First, let the polygon be parameterized by (u, v) so that the position vector from the polyhedron's center of volume to a point on the polygon is:

$$\vec{r} = \langle r_{0,x} - r_{c,x} + u\hat{u}_x + v\hat{v}_x, r_{0,y} - r_{c,y} + u\hat{u}_y + v\hat{v}_y, r_{0,z} - r_{c,z} + u\hat{u}_z + v\hat{v}_z \rangle \tag{7.19}$$

where $r_0 = (r_{0,x}, r_{0,y}, r_{0,z})$ is the 3D point in global coordinates representing the origin of the polygon's local (u, v) coordinate system. Point $r_c = (r_{c,x}, r_{c,y}, r_{c,z})$ is the agent polyhedron's center of volume. Unit vectors $\hat{u} = \langle \hat{u}_x, \hat{u}_y, \hat{u}_z \rangle$ and $\hat{v} = \langle \hat{v}_x, \hat{v}_y, \hat{v}_z \rangle$ are bases for the (u, v) coordinate system, expressed in global coordinates. Using this position vector, the surface integral can be written:

$$\left.\frac{dV}{d\psi}\right|_i = \int\int \left(\hat{\psi} \times \vec{r}\right) |\vec{r}_u \times \ \vec{r}_v| du dv \tag{7.20}$$

where the unit vector $\hat{\psi}$ points in the direction of $\dot{\psi}$. The cross product $\hat{\psi} \times \vec{r}$ represents the amount of change at point (u, v), where this change is integrated over the polygon. The second cross product $|\vec{r}_u \times \ \vec{r}_v|$ is due to the parameterization of S with (u, v), where $\vec{r}_u$ and $\vec{r}_v$ are the partial derivatives of $\vec{r}$ with respect to u and v. To simplify the limits of integration, the interface polygons are decomposed into triangles such that (7.20) becomes:

$$\left.\frac{dV}{d\psi}\right|_i = \int_0^{v_1} \int_0^{mv+b} \left(\hat{\psi} \times \vec{r}\right) |\vec{r}_u \times \ \vec{r}_v| du dv \tag{7.21}$$

where v_1, m, and b are constant geometric properties of the triangle, i.e. (u, v) are integrated over the regions: $0 \leq u \leq mv + b$ and $0 \leq v \leq v_1$.

Eq. (7.19) can be plugged into (7.21), which produces a double integral in (u, v) and is straightforward to evaluate. Integrals $dV/d\theta$ and $dV/d\phi$ are similar to compute. One difference is the direction of the rotation vector $\hat{\psi}$, $\hat{\theta}$, and $\hat{\phi}$. Because the polyhedron undergoes successive Euler rotations, computation of this vector depends on the order of rotations through a rotation matrix. In order to ensure the gradients are computed quickly, the integral can be symbolically computed beforehand such that the program must only plug in parameters at runtime.

7.5 Approximating Trajectory Risk

While $\mathcal{W}_{int}$ serves as a measure of collision constraint violation, it must be integrated with a sampling routine to approximate trajectory risk. A natural way to approximate the collision probability is to sample a large number of scenarios simulating the agent's trajectory, recording which trajectories are in collision, and using the frequency of collision as an estimate. Unfortunately, $\mathcal{W}_{int}$ cannot be input into this estimate directly because $|\mathcal{W}_{int}| \in \mathbb{R}$ where $|\mathcal{W}_{int}| \geq 0$, i.e. it would not provide a valid probability. To do this, inspiration is taken from the particle filter approach in [14], where binary decision variables are used as indicator functions to model whether a scenario is in collision. Specifically, the collision probability is estimated as:

$$P\left(\text{Collision}\right) = \frac{1}{S} \sum_{s} \mathbb{1}_{\text{Collision}} \tag{7.22}$$

where s is a sampled scenario. Given computation of $\mathcal{W}_{int}$, the collision probability is approximated using indicator functions as:

$$P\left(\text{Collision}\right) \approx \frac{1}{S} \sum_{s} \mathbb{1}_{|\mathcal{W}_{int}|>0} \tag{7.23}$$

However, as mentioned in Section 7.3, Eq. (7.23) lacks a non-zero gradient over a larger region, which would not work well with an SQP optimizer. Instead, this thesis approximates the indicator variable through a sigmoid function. The sigmoid

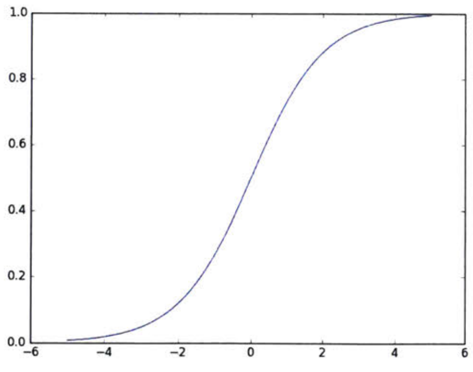


Figure 7-10: The shape of a general sigmoid function

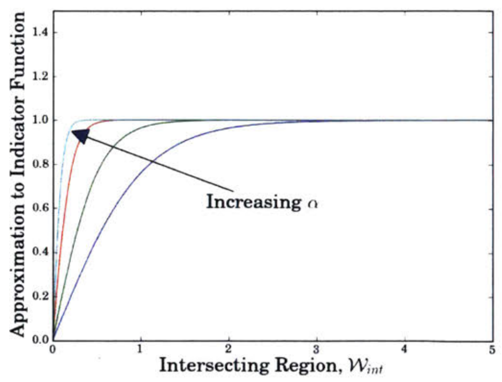


Figure 7-11: The modification of the sigmoid function used to approximate $\mathbb{1}_{|\mathcal{W}_{int}|>0}$. Increasing α improves the approximation.

has seen prior use in non-convex optimization problems to approximate indicator or barrier functions [35]. The general form of the sigmoid is:

$$\mathrm{Sig}_{\alpha}(s) = \frac{1}{1+\exp(-\alpha s)} \tag{7.24}$$

where α is a parameter that dictates the steepness of the slope. A plot of this function is shown in Fig. 7-10. From the figure, it's apparent how this may model an indicator variable; negative arguments mode the case $\mathbb{1} = 0$ and positive arguments model the case $\mathbb{1} = 1$. The sigmoid becomes a better approximation to the indicator function as α grows large.

Eq. (7.24) must be modified to model the collision constraint. The intersecting region cannot be negative and the function should be zero at $|\mathcal{W}_{int}| = 0$. This suggests using a sigmoid of the form:

$$\mathrm{Sig}_{a}(s) = C\left(\frac{1}{1+\exp(-\alpha s - B)} - A\right) \tag{7.25}$$

where A, B, and C are constants that provide the required offset and scaling. This function is shown in Fig. 7-11. The indicator function of Eq. (7.23) is approximated as:

$$\mathbb{1}_{|\mathcal{W}_{int}|>0} \approx \text{Sig}_\alpha\left(|\mathcal{W}_{int}|\right) = \frac{1}{1+\exp\left(-\alpha|\mathcal{W}_{int}| - B\right)} - A \tag{7.26}$$

and the collision probability approximated as:

$$P\left(\text{Collision}\right) \approx \frac{1}{S}\sum_s \text{Sig}_\alpha\left(|\mathcal{W}_{int,s}|\right) \tag{7.27}$$

The overall chance constraint F_{cc} is written:

$$\epsilon - \frac{1}{S}\sum_s \text{Sig}_\alpha\left(|\mathcal{W}_{int,s}|\right) \geq 0 \tag{7.28}$$

An advantage of the sigmoid function is that it's easy to differentiate when computing the Jacobian matrix. Specifically,

$$\frac{d\text{Sig}_\alpha\left(s\right)}{ds} = \text{Sig}_\alpha\left(s\right)\left(1 - \text{Sig}_\alpha\left(s\right)\right) \tag{7.29}$$

The chain rule is used to derive the gradients with respect to the configuration states:

$$\frac{\partial F_{cc}}{\partial x_n} = \frac{d\text{Sig}_\alpha\left(\mathcal{W}_{int}\right)}{d\mathcal{W}_{int}}\frac{\partial \mathcal{W}_{int}}{\partial x_n} \tag{7.30}$$

with similar expressions for the other configuration states.

One downside to the sigmoid approximation is tuning α. As shown in Fig. 7-11, increasing α enables a better approximation to the collision check. However, this also makes the derivative nonzero over a smaller region and makes it more difficult to integrate into a sequential quadratic program. An α that is too large typically results in failure of the SQP solver. For this thesis, α was tuned by hand depending on the problem. The issue of convergence and tuning α is discussed further in the following chapter.

7.6 Integration with the Hybrid Search's Trajectory Optimization

The collision constraint discussed in this chapter is integrated into the lower level motion planner of the hybrid search. The motion planning component is framed as a problem in optimal control and solved as a trajectory optimization. This section explains the trajectory optimization.

In general, trajectory optimization is an approach to solve optimal control problems where an agent must navigate from an initial state to a goal state while constrained by its dynamics and some objective function is minimized. While analytical methods exist to solve optimal control problems, such as using Pontryagin's Minimum Principle, these can be difficult to apply to more complex problems [9]. All of the trajectory optimization approaches in this thesis focus on *transcription* methods. Transcription refers to the process of transforming the optimal control problem into a mathematical program. If the dynamics are represented by nonlinear equations of motion, the resulting mathematical program is a nonlinear program. An advantage of transcription methods versus analytic methods is that they can handle very complex dynamical systems, goals, and constraints. Furthermore, a wide range of off-the-shelf optimization libraries exist that can compute solutions quickly.

Within transcription approaches, there are two main types of trajectory optimization: direct collocation and shooting methods. This chapter describes a method to use risk-bound motion planning with a collocation method; the next chapter describes Shooting Method Monte Carlo, which is based on a shooting method. For general trajectory optimization using transcription, an excellent set of introductions and tutorials is provided in [9], [59], and [60].

7.6.1 Assumptions Regarding Uncertainty

The approach presented in this chapter approximates the chance constraint by sampling from a state distribution and computing the risk with respect to the samples.

This raises the question of which distribution is used for sampling. In this chapter, the assumption is made that uncertainty is additive Gaussian noise. This means that state uncertainty is approximated by adding a sample to each state:

$$\mathbf{x}_s = \mathbf{x} + \mathbf{Z}_s \;\; \forall \;\; s \in S \tag{7.31}$$

where $\mathbf{x}$ is the vector of state decision variables and $\mathbf{Z}_s$ is a vector of sampled random variables for scenario s. Importantly, the samples are drawn prior to the optimization and remain unchanged for each scenario throughout the optimization. This helps the trajectory optimizer converge; if states were re-sampled at each iteration of the trajectory optimization, the gradient of the collision constraint with respect to the decision variable would be much more complicated. This assumption means that uncertainty is independent of state.

While additive Gaussian noise provides a quick approximation to state uncertainty, it is inaccurate for nonlinear systems such as the Dubins car and Slocum glider. A more realistic model is discussed in the next chapter that does not rely on the Gaussian assumption.

7.6.2 The Direct Collocation Method

The direct collocation method is characterized by interpolating the trajectory and using the interpolation knot points (or *collocation* points) as decision variables in the resulting optimization. A wide variety of numerical integration methods are possible depending on the desired accuracy of the interpolation. Typical numerical integration schemes are the trapezoidal method and Hermite-Simpson interpolation.

In general, it is simpler to implement than the shooting method. This is because nonlinear optimization codes (such as sequential quadratic programs) typically require analytical evaluation of the constraint gradients to perform efficiently. Computing the Jacobian matrix for the collocation method only requires consideration of the surrounding time steps where the exact number depends on the order of the integration method.

Notation Regarding Time

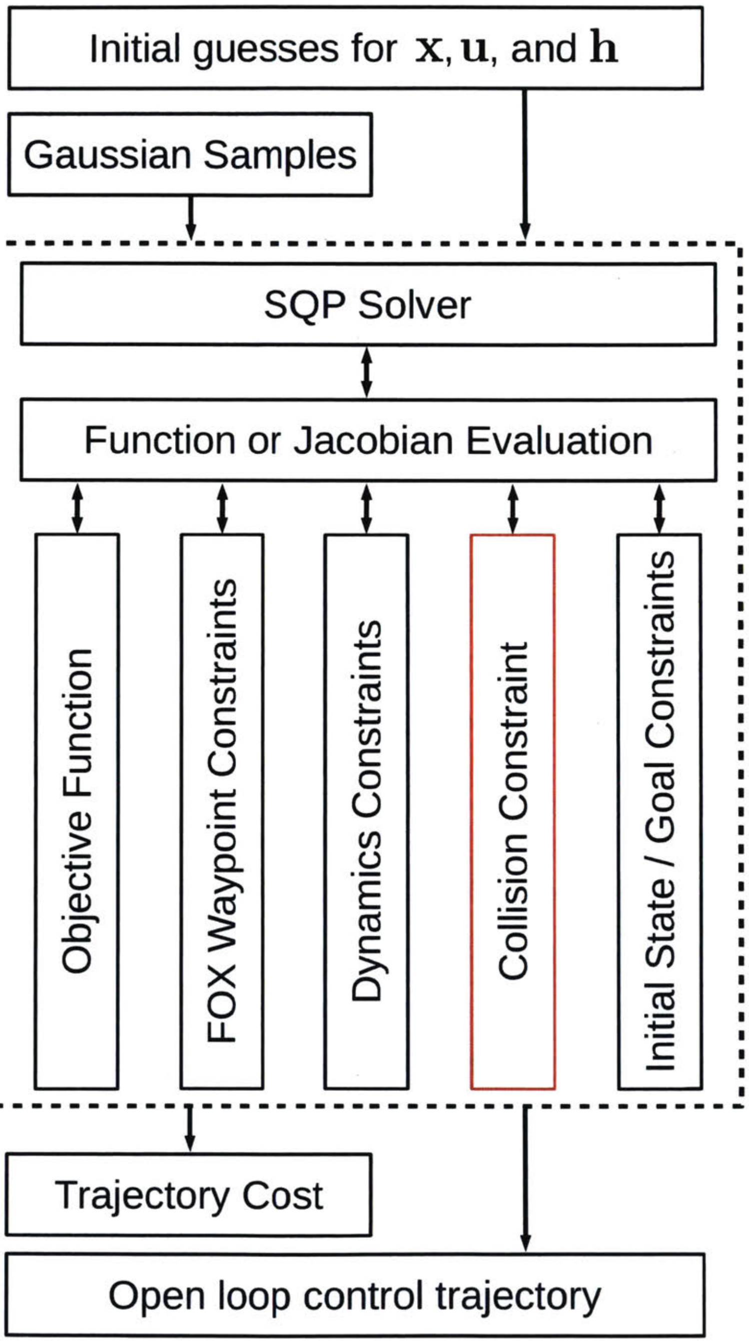
Similar to earlier chapters, the variable t is used to denote continuous time, i.e. $t \in \mathbb{R}$. Variable k is used to denote collocation knots in a trajectory optimization: x_k represents the state x at knot k. The change in time between different collocation knots is h, i.e. $t_{k+1} - t_k = h_k$. Each h_k is a decision variable, which is solved by the optimizer. From Chapter 4, variable n indexes one of FOX's waypoints. Uppercase K and N denote the total number of knot points and waypoints respectively where typically $N << K$. The number of knot points between each waypoint is an input and held constant.

7.6.3 Trajectory Optimization via Direct Collocation

This section describes how this chapter's chance constraint formulation is integrated into a direct collocation trajectory optimization. The trajectory optimization receives waypoint constraints, initial state, and goal states from FOX. Samples that describe the agent's state distribution are generated as well as initial guesses for the decision variables. The decision variables are $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{h}$, representing the states, control inputs, and time steps between collocation knots. There are a fixed number of collocation knots between waypoints but each individual time step h may vary as the optimizer requires. After solving the nonlinear program, the output is an open loop trajectory $\mathbf{u}_{ol}$ and path cost f_0. The path cost is used by FOX to compare against other paths in the upper level region planner. There are six types of functions that must be evaluated at each iteration of the SQP solver: the objective function, constraints on the dynamics, FOX's waypoint constraints, the collision constraint, and initial state and goal constraints. An overview of the risk-bound trajectory optimization using direct collocation is shown in Fig. 7-12.

Objective Function

A wide variety of objective functions are possible including minimizing the distance traveled, minimizing time, or minimizing control input. A simple objective minimizing



Initial guesses for $\mathbf{x}, \mathbf{u}$, and $\mathbf{h}$
Gaussian Samples
SQP Solver
Function or Jacobian Evaluation
Objective Function
FOX Waypoint Constraints
Dynamics Constraints
Collision Constraint
Initial State / Goal Constraints
Trajectory Cost
Open loop control trajectory

Figure 7-12: An overview of the trajectory optimization inputs, outputs, and function evaluations. The box in red relates to the sampling-based chance constraint described in this chapter. It can be parallelized on a graphics processing unit (GPU).

the distance traveled, in the $\mathcal{W}^2$ case, is:

$$f = \sum_{k}^{K-1} \sqrt{\left(x_{k+1} - x_k\right)^2 + \left(y_{k+1} - y_k\right)^2} \tag{7.32}$$

with an added z state for the $\mathcal{W}^3$ case.

Dynamical Constraints

The solution to the optimal control problem must respect the dynamics constraints. Part of the challenge of transcribing the optimal control problem as a nonlinear optimization problem is transforming the dynamics from continuous time to a finite number of variables and constraints. This involves approximating the states derivatives $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ with a numerical quadrature rule. To do this, there are many possibilities depending on the accuracy and complexity of the desired approximation. The simplest possible approximation is the Euler method:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{h} = f(\mathbf{x}, \mathbf{u}) \tag{7.33}$$

Because of its simplicity, Euler's method has been widely used in previous motion planning under uncertainty and motion planning algorithms [38], [80], [15]. Unfortunately, Euler's method is order h, which means its error scales as a function of the time step. This is relatively poor and typically higher order approximations are used.

This work uses the trapezoidal method to enforce the dynamics constraints. This method relies on the trapezoidal rule to carry out integrations:

$$x_{k+1} - x_k \approx \frac{1}{2} h_k \left(f\left(x_{k+1}, u_{k+1}\right) + f\left(x_k, u_k\right)\right) \tag{7.34}$$

Because both the left and right-hand sides are defined in terms of x_{k+1} and it is generally difficult to solve for x_{k+1}, the trapezoidal method is *implicit.* The trapezoidal rule has error of order h^2. In general, reducing h by a factor of two will reduce the error by a factor of four.

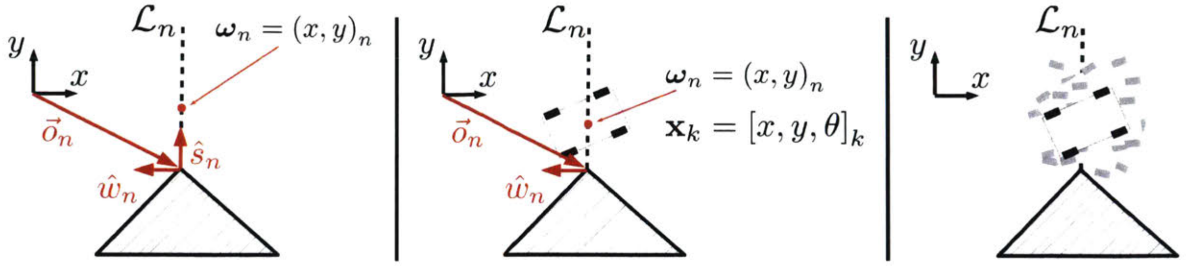


Figure 7-13: The above three panes illustrate how agents with geometry are integrated into FOX's landmark regions $\mathcal{L}_n$ presented in Chapter 4. The pane on the left depicts the constraint plane with a point robot as described in Chapter 4; the point robot was constrained to lie in the plane $\hat{w}_n = 0$ and the risk evaluated as a function of the point's distance along the axis $\hat{s}_n$ as described in Chapter 6. The middle pane shows how waypoints are handled for agents with geometry; only the states (x, y) are relevant to determining whether the agent lies on the ray $\mathcal{L}_n$. A similar construction holds for 3D workspaces where the relevant stats are (x, y, z). For this construction, the $\hat{s}_n$ axis is no longer required because the risk is computed via sampling rather than an evaluation of a CDF. This is illustrated in the pane on the right; samples are generated using the decision variables at the k^{th} collocation knot.

Waypoint Constraints

Waypoint constraints are passed from FOX to the trajectory optimization routine. Unlike in Chapter 6, axes $\mathbf{s}$ and $\mathbf{r}$ are not needed. This is because the chance constraint is approximated via sampling rather than CDFs. Only the $\mathbf{w}$ constraint is necessary to ensure the state remains on the waypoint. The geometric constraints are illustrated in Fig. 7-13.

Probabilistic Collision Constraint

The probabilistic constraint relies on Eqn. (7.28). As described in Subsection 7.6.1, state uncertainty is approximated via sampling from a Gaussian distribution and adding this uncertainty to the state at each knot point, $\mathbf{x}_k$.

To be specific, if the optimization subroutine's best guess of the decision variables at iteration i is $\mathbf{x}_{k,i}$, a Gaussian sample is added to each configuration state as in Eq. (7.31):

$$x_{k,i,s} = x_{k,i} + z \;\; \forall s \in S \tag{7.35}$$

where S is the total number of samples and $z \propto \mathcal{N}(\mu, \sigma^2)$. The intersecting area

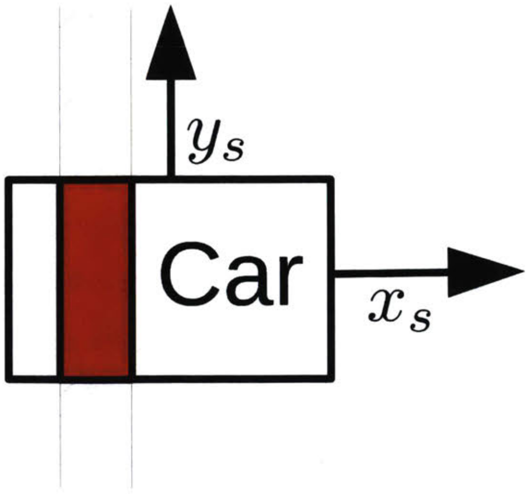


Figure 7-14: A poor initialization of an optimization routine, which is likely to fail. The hatched obstacle intersects the midpoint of the car; the intersection is colored red. The gradients are unlikely to "push" the optimizer to a feasible path.

between the agent and obstacles are evaluated with respect to these samples and this enables approximation of the chance constraint:

$$P\left(\text{Failure}\right) \approx \frac{1}{S}\sum_{s} \text{Sig}_{\alpha}\left(\left|\mathcal{W}_{int,s}\left(x_{k,i,s}\right)\right|\right) \tag{7.36}$$

where $x_{k,i,s}$ is meant to be the configuration states at optimization knot k at optimization iteration i for sample s.

Initial State and Goal Constraints

The initial and goal constraints are input as constants; the initial and final state decision variables are set equal to them:

$$\mathbf{x}_0 = \mathbf{x}_i \tag{7.37}$$

$$\mathbf{x}_K = \mathbf{x}_g \tag{7.38}$$

7.6.4 Importance of Initial Guesses

In general, SQP methods are greatly affected by the initial guess. Oftentimes, the optimization routine will not succeed if the initial guess is not feasible. In the obstacle-avoidance use case, infeasible initial guesses can easily lead to the optimizer either failing to converge to a solution or returning erroneous results. For example, the image in Fig. 7-14 presents a case when the optimizer is initialized with an obstacle through the midpoint of the Dubins car. This case is likely to cause the optimizer to fail or produce an infeasible trajectory because the gradients are unable to determine how to restore feasibility. For this reason, our implementation attempts to initialize an initial feasible guess.

7.6.5 Parallelization on GPUs

Recently, advancements in graphical processing units (GPUs) have helped algorithms scale to real world problems. The massive parallelization provided by GPUs has generated huge advancements in the learning community. They also provide a benefit to motion planning algorithms and algorithms that must deal with uncertainty. Other work has investigated this idea as well for the stochastic and deterministic path planning case [48, 5].

There are two areas where GPUs offer considerable benefit for risk-bound motion planning: constraint evaluation and computing stochastic quantities. Regarding constraint evaluation, the insight is that motion planning problems often have large numbers of constraints; constraints must be typically evaluated at every time step where T can grow quite large. In many cases, the evaluation of individual constraints is non-trivial. This is the case with collision checks. It is observed in [5] that the probabilistic road map (PRM) algorithm is massively parallelizable.

GPUs can also help evaluate the stochastic model. Planning under uncertainty is hard because computing moments of complex distributions is often intractable. The complex integrals can be approximated with numerical quadrature methods, where

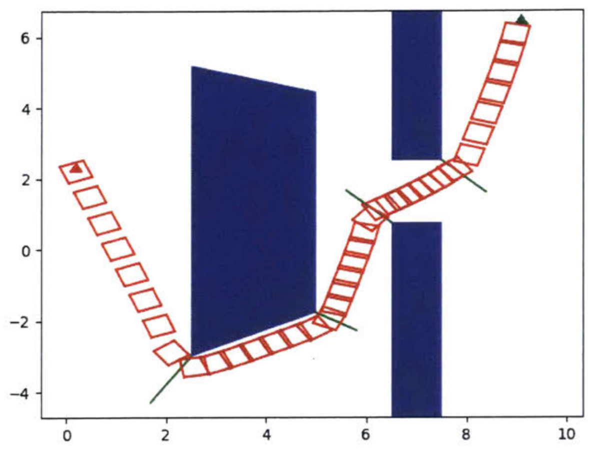


Figure 7-15: Example illustrating non-holonomic dynamics with a simple Dubins car model. Green segments are landmark regions.

the general form is:

$$\int f(x)\,dx \approx \sum_{i}^{n} w_i f(x_i) \tag{7.39}$$

where the right hand side evaluates the function f at discrete points and multiplies the evaluations by weights w. The insight is that these function evaluations can often be costly to evaluate but the right-hand side can oftentimes be trivially parallelized. For the algorithms discussed in this chapter, the sampling-based collision checks are trivially parallelized. The parallelized version shows a considerable speedup compared with the serial version.

7.7 Results

Test cases were run on the Dubins car and Slocum glider. Comparisons were performed to illustrate the model's ability to meet the chance constraint in simulation and the advantages of parallelization.

The Slocum glider was tasked with exploring a region of the Kolumbo volcano's caldera, as described in Chapter 5. Instead of the modified linear dynamical model used in the previous chapter, this model relies on the full nonlinear model from Section 7.2.2. Because the nonlinear model is considerably more complicated, solving times for

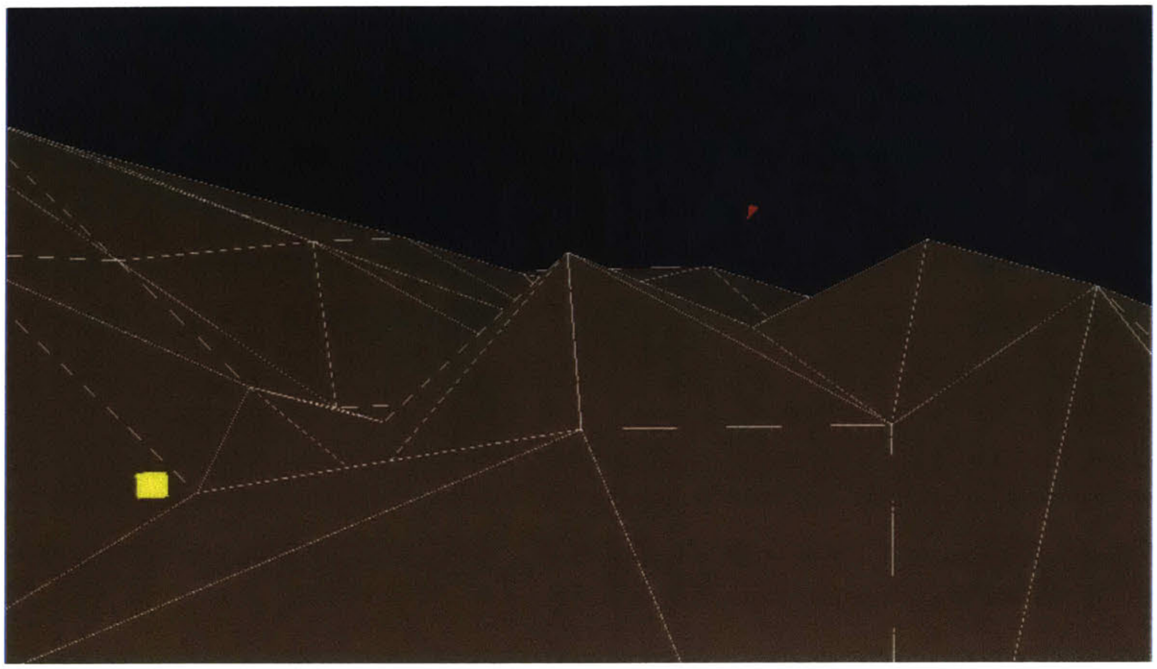


Figure 7-16: The glider was tasked with exploring the ridge at the caldera's northeast corner. The glider was required to navigate through a relatively tight valley from the red triangle to the yellow square.

the trajectory planner are longer. Therefore, the glider is only tasked with exploring a region of the caldera, rather than the entire caldera as in Chapter 5. The test case is shown in Fig. 7-16 where the glider must navigate a valley at the caldera's northeast corner. This region of the caldera is of interest because of undersea thermal vents. The glider must navigate this region while maintaining a bound on the probability of collision; the mission is made more difficult by unknown ocean currents.

Example trajectories are shown in Figs 7-17 and 7-18. Fig. 7-17 presents a trajectory where there is little uncertainty that the ocean currents are small. In this case, a multivariate Gaussian was added to the configuration states with $\boldsymbol{\mu} = [0, 0, 0, 0, 0, 0]$ and $\sigma_x = 0.05$, $\sigma_y = 0.05$, $\sigma_z = 0.05$, $\sigma_\psi = 0$, $\sigma_\theta = 0$, $\sigma_\phi = 0$ (all off-diagonal terms in the covariance matrix were assumed zero). As shown in Fig. 7-17, the small amount of uncertainty means the optimizer can place the trajectory very close to the ocean surface; the generated trajectory is smooth with a slight curve to compensate for the ridge. The second example, Fig 7-18, presents a trajectory generated under more uncertainty in the ocean currents. For this example, $\boldsymbol{\mu} =$

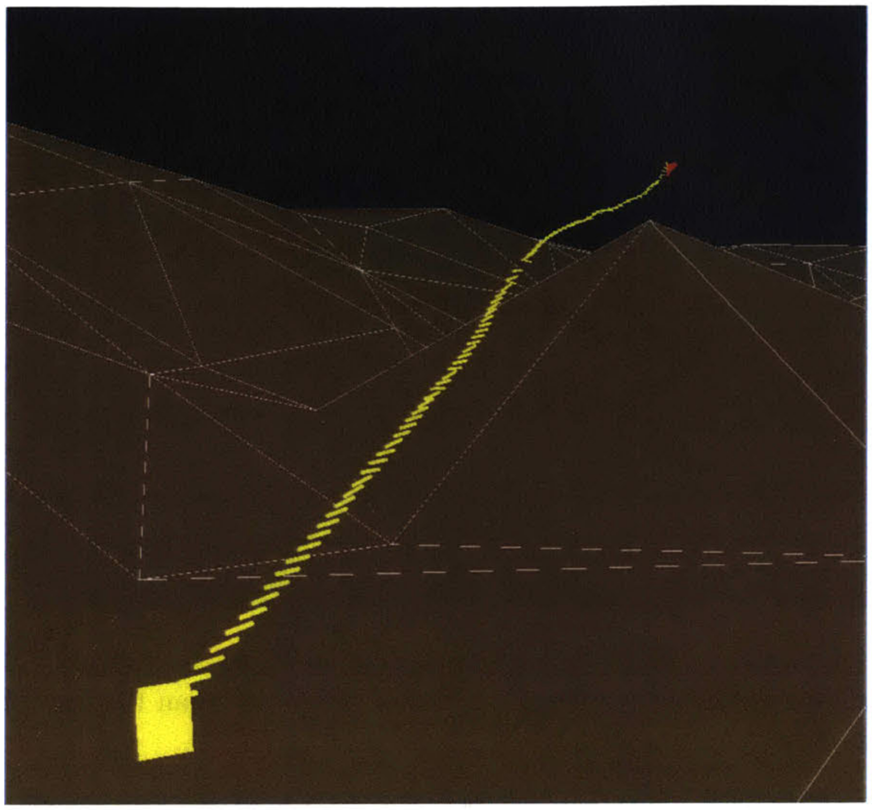


Figure 7-17: A trajectory under the assumption of small ocean currents and little uncertainty about the currents.

$[-5.0, 0, -5.0, 0, 0, 0]$ and $\sigma_x = 5.0$, $\sigma_y = 0$, $\sigma_z = 5.0$, $\sigma_\psi = 0$, $\sigma_\theta = 0$, $\sigma_\phi = 0$ (once again, all off-diagonal terms in the covariance matrix were zero). This might represent a situation where there is an uncertain current pushing the glider in the negative x and negative z directions. As depicted in Fig. 7-18, this changes the output trajectory considerably. The optimizer produces a solution where the glider is kept at approximately the same depth until abruptly descending after passing the ridge.

Both Figs 7-17 and 7-18 show the nominal glider trajectory, i.e. they plot the knot points of the collocation trajectory optimization. As described in this chapter, the chance constraint is computed via sampling from distributions describing the configuration states and adding the samples to the collocation decision variables. A plot of these samples is shown in Fig. 7-19. This test case was for 100 samples per collocation step; for clarity, only every third sample is shown in the figure.

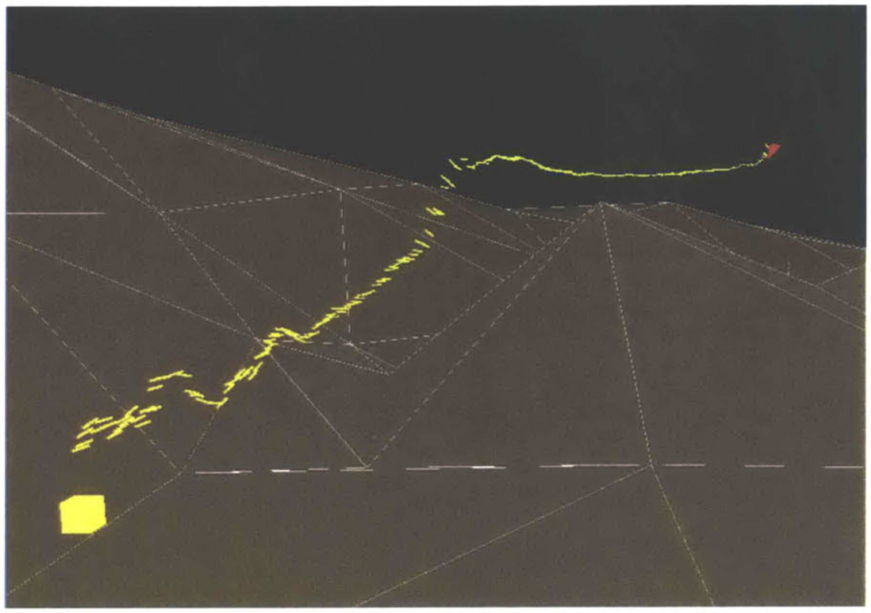


Figure 7-18: A trajectory under larger uncertainty in the ocean currents. The trajectory optimizer generates a significantly different trajectory than the case of the case with low uncertainty.

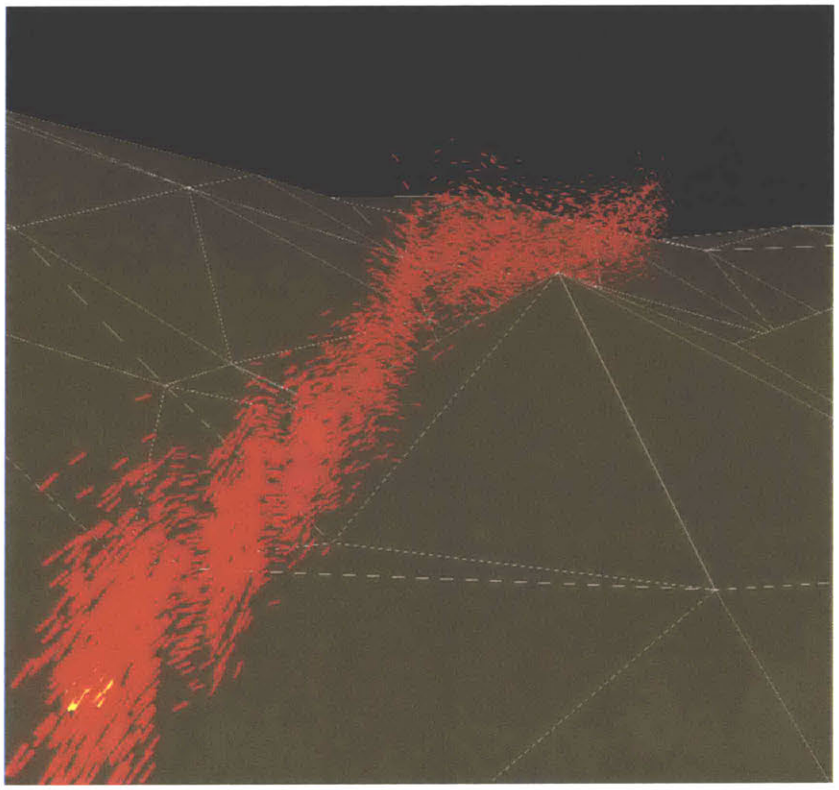


Figure 7-19: Depiction of the samples used to compute the trajectory risk.

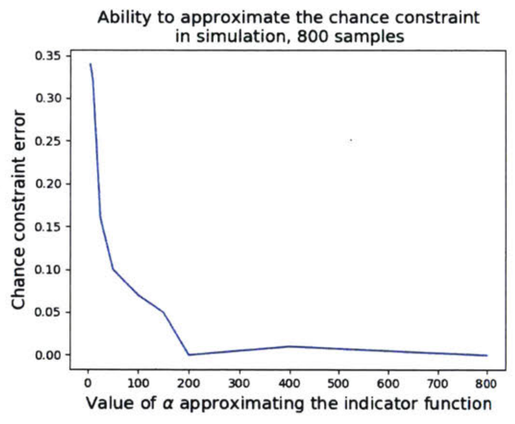


Figure 7-20: The ability to meet the chance constraint in simulation as a function of α

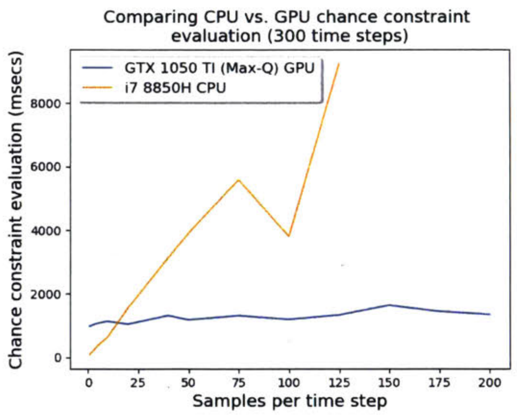


Figure 7-21: Comparison of using sampling to approximate a chance constraint on a CPU vs. a GPU

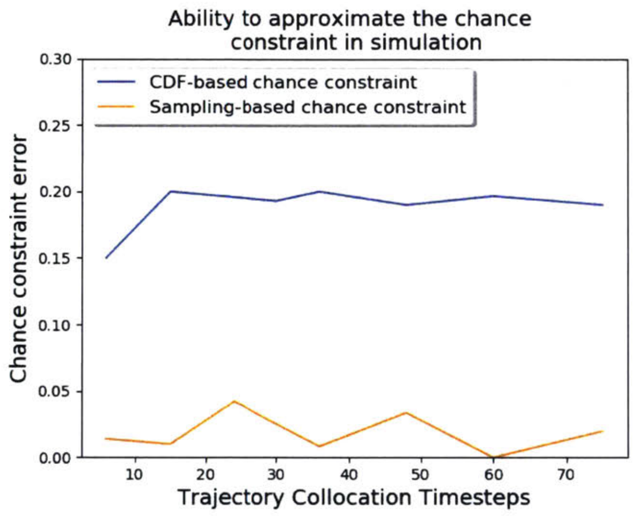


Figure 7-22: Error of a CDF-based chance constraint model vs. a sampling-based model in simulation. The sampling-based model assumed $\alpha = 200$ and used 800 scenarios.

Numerical results are presented in Figs. 7-20 - 7-22. Figure 7-20 presents the ability of the sampling-based chance constraint model to approximate the trajectory risk in simulation as a function of the α parameter of Eq. (7.27). Recall that the sampling based chance constraint is approximated as

$$P\left(\text{Collision}\right) \approx \frac{1}{S} \sum_{s} \text{Sig}_{\alpha}\left(\left|\mathcal{W}_{int,s}\right|\right) \tag{7.40}$$

where $\mathcal{W}_{int,s}$ is the area/volume of intersection between the agent and obstacle in scenario s. This function approximates the indicator function of whether a particular sample is in collision; the approximation improves as α increases. This is depicted in Fig. 7-20. A significant advantage of sampling-based methods to approximate trajectory risk is that they are straightforward to parallelize. Collision checks and the corresponding Jacobians can be parallelized across the threads of a GPU. The improvement is significant as shown in Fig. 7-21, which compares the glider ridge-navigation computation on a GPU vs. a CPU. There is a cost to transferring data to the GPU; instances with fewer samples per time step perform better on a CPU. However, as the number of scenarios becomes large, there is a clear advantage to the GPU implementation.

Figure 7-22 shows the advantage of using a sampling-based chance constraint model versus a CDF-based model. Both models assume the same nonlinear glider dynamics. The CDF-based model assumes the glider's geometry is concentrated at a single point and the methods of Chapter 6 used to compute trajectory risk. The sampling-based model uses the method in Section 7.5. The risk of both trajectories is then simulated as follows. Given the trajectory $\mathcal{T}$ and collocation knot k, denote the nominal configuration state at k as $\mathcal{C}_k$. In the glider use case $\mathcal{C}_k = (x, y, z, \psi, \theta, \phi)_k$. For each configuration state, a sample is drawn from the distribution describing the agent's state uncertainty and added to $\mathcal{C}_k$. Let this new configuration be $\mathcal{C}'_k$, e.g., $\mathcal{C}'_k = (x + \Delta x, y + \Delta y, z + \Delta z, \psi + \Delta\psi, \theta + \Delta\theta, \phi + \Delta\phi)_k$. Each $\mathcal{C}'_k$ is associated with a scenario, $s \in S$. A large number of scenarios are simulated; the number of scenarios divided by the total number of scenarios determines the simulated trajectory risk. The error in Fig. 7-22 is defined as the difference between the value of the chance constraint per the trajectory optimization and the value computed through simulation. As is shown in the figure, the sampling-based chance constraint matches the trajectory risk in simulation much better than the CDF-based model. For all tests, the chance constraint value was $\epsilon = 0.20$. Typically, the CDF-based model output a trajectory with an active chance constraint (i.e. $P\left(\text{Fail}\right) = \epsilon$), but the simulated trajectory had little risk.

7.8 Conclusion

This chapter extends the approaches in previous chapters to approximate risk for agents with nontrivial geometric dimensions. To do this, the key insight is that the probabilistic problem is difficult to solve via integrating over all agent configurations and a sampling-based method is proposed. The sampling-based method has the advantages of relying on fast geometric subroutines and can be parallelized for additional speed. This chapter also presented how this method may be integrated into a collocation trajectory optimization for integration with the hybrid search's trajectory planner.

One remaining question is how well the sampling-based method can approximate the true collision risk. It is difficult to provide an answer to this given only the techniques discussed in this chapter; the state distribution was approximated via sampling from a Gaussian and, for nonlinear dynamical systems, the state distribution is rarely Gaussian. I propose a more accurate approximation to trajectory risk in the next chapter in presenting the Shooting Method Monte Carlo algorithm.

Chapter 8

Shooting Method Monte Carlo

The previous chapter presented an approach for combining risk-aware motion planning and agents with nontrivial geometries. The insight to computing risk was to transform an intractable integration into a simpler geometric problem. The geometric problem required sampling from the agent's state and determining which samples are in collision. By the law of large numbers, the estimate of the risk will converge to the true risk as the number of samples grows large.

An unanswered question is from what distribution the samples are drawn to compute the trajectory's risk. The dynamical models considered in the previous chapter are nonlinear. Due to the non-linearity, the state uncertainty is non-Gaussian, even if the underlying uncertainty model is additive Gaussian noise. Therefore, it is inaccurate to simply sample the state from a Gaussian distribution. To account for more complex distributions, this chapter introduces Shooting Method Monte Carlo (SMMC). Essentially, this approach combines Monte Carlo simulation with a shooting method trajectory optimization to allow for motion planning under uncertainty with more complex dynamics and models of uncertainty.

The model of time also plays an important role when planning for real world systems. All motion planning under uncertainty algorithms ultimately model systems that are based in continuous time. Often they are described by nonlinear differential equations which must be integrated numerically. The integration methods for such systems often require precise sizing of the time step. However, most methods for path planning under uncertainty use discrete time models. These discrete time models must be adapted to model a real world continuous time system. Rather than rely on a discrete time stochastic process, I begin this chapter by introducing a method to integrate a continuous time stochastic process with SMMC.

This chapter proceeds as follows. The first section discusses the discrete-time linear model often used in literature for planning under uncertainty. Second, I explain the continuous time stochasticity used by the Shooting Method Monte Carlo algorithm. The next section concerns the SMMC algorithm itself followed by an example computation of the objective, constraints, and Jacobian matrix.

8.1 Standard Approach: Discrete Time Linear Dynamics with Additive Gaussian Noise

As mentioned earlier in this thesis, much of the prior work on path and motion planning under uncertainty has focused on approximations that make the problem tractable. Some of the key assumptions are that the dynamics are linear time invariant and stochasticity is additive Gaussian noise [12], [80]. A typical form of the optimal control problem is:

$$\begin{aligned} \text{mininimze} \quad & C\mathbf{x} && (8.1)\\ \text{subject to} \quad & \mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \mathbf{w} && (8.2)\\ & \mathbf{x}_0 = \mathcal{I} && (8.3)\\ & \mathbf{x}_T = \mathcal{G} && (8.4) \end{aligned}$$

where $\mathbf{w} \sim \mathcal{N}(0, \Sigma)$. This formulation of the optimal control problem makes the problem much more tractable for two reasons. First, the assumption of Gaussian uncertainty allows propagation of the distribution's statistics in time. This is because the addition of two Gaussian random variables is still Gaussian. Namely, given $X \sim$

$\mathcal{N}(\mu_x, \sigma_x^2)$, $Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$ then $Z = X + Y$ implies $Z \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$. This idea can be extended to Eq. 8.2, which means the distribution of state $\mathbf{x}$ is known at any future time and is Gaussian, as shown in [15]:

$$\mu_t = \sum_{i=0}^{t-1} A^{t-i-1} B\mathbf{u}_i + A^t \hat{\mathbf{x}}_0 \tag{8.5}$$

$$\Sigma_t = \sum_{i=0}^{t-1} A^i Q \left(A^T\right)^i + A^t P_0 \left(A^T\right)^t \tag{8.6}$$

where the initial state has distribution $\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, P_0)$ and Q is a covariance matrix. Because the state distribution is known at all future time steps, it is relatively straightforward to compute the chance constraint via the Gaussian cumulative distribution function. Although the Gaussian CDF is not analytic, many fast numerical approximations have been developed for the single and multivariate case. Most modern numerical software libraries include a routine to compute the single variate CDF; work due to [36] is available in C and Python.

For many planning problems, linear approximations to the dynamics suffice. However, if a nonlinear dynamical model is required, it's no longer accurate to use a method based on the Gaussian CDF. Even in the case of additive Gaussian noise, the state distribution is no longer necessarily Gaussian and is likely to be a complex distribution, intractable for evaluating the integrals necessary for planning under uncertainty. Because of the difficulty of propagating the state distribution in the nonlinear case, the remainder of this chapter focuses on a Monte Carlo approach.

8.2 Connecting Continuous Time Dynamics with Continuous Time Uncertainty

One of the simplifications of the linear model presented in the previous section is that it relies on a discrete-time stochastic model. This is disadvantageous for a couple of reasons. First, real world dynamical systems are described by continuous time differential equations. Using a discrete time stochastic process inherently requires

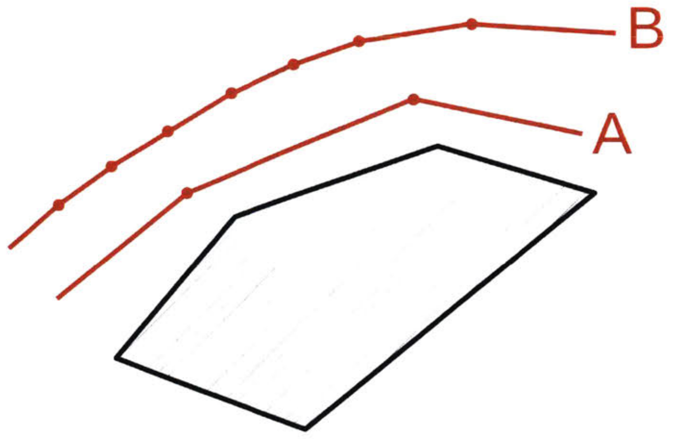


Figure 8-1: Illustration of how trajectories can become more conservative simply by decreasing time step size when the stochastic process variance is held constant. The diagram assumes the probability of failure is evaluated only at waypoints (red dots). For the same variance, trajectory A has fewer waypoints (a larger time step) and therefore fewer evaluations of path risk. This enables the optimizer to bring it closer to the obstacle while maintaining the overall chance constraint.

some form of time approximation / discretization. Second, many of the numerical integration routines for nonlinear systems require precise specification of the time step (typically, smaller time steps improve the accuracy of the integration). Additionally, many trajectory optimization routines (this work included) make the time step a decision variable that is changed by the optimizer. These issues raise the question of how to vary the stochastic model as the time step is changed for models that rely on a discrete time stochastic process. The simplest approach is to assume the stochastic model has a constant variance, i.e. the variance is not a function of the time step. This approach has the problem that as Δt decreases, the trajectory becomes more conservative. This problem is illustrated in Fig. 8-1.

To solve these issues, I propose using a continuous time model of uncertainty. This avoids the unnecessary discretization of a real world continuous time system and generates a model where the variance of the stochastic process is a function of the time step. It is challenging to combine a continuous time stochastic model with trajectory optimization because the transcription of constraints into a mathematical program requires some form of discretization. This is not unique to trajectory optimization approaches; most path and motion planning algorithms involve some form of implicit discretization, especially when obstacle avoidance is considered. Therefore, not only is it necessary to generate a continuous time stochastic model but also one that can be numerically integrated and used with a motion planning algorithm.

The continuous time stochastic model used in this chapter is a Wiener process. A Wiener process (Brownian motion) is a continuous time stochastic process that can be thought of as the limit of a random walk as the step size becomes small [64]. In the context of this chapter, the Wiener process is useful because it typically serves as the source of randomness in a stochastic differential equation (SDE). This SDE can be used to model the uncertain evolution in time of a dynamical system. A Wiener process W is defined by three properties: $W_0 = 0$, W as a function of time t is continuous with probability 1, and W has independent increments such that $W_{t+s} - W_s \sim \mathcal{N}(0, t)$ [21].

The general form of a stochastic differential equation (SDE) written in differential form is:

$$dX(t) = f(t, X(t)) + B(t, X(t))\, dW(t) \tag{8.7}$$

where X is a random variable and $dW(t)$ represents a Wiener process. Variable B represents a drift coefficient. For this thesis, only drift terms are considered where $B(t, X(t))$ is constant. Eq. (8.7) is written only symbolically to mean the integral equation [64]:

$$X_t = X_{t_0} + \int_{t_0}^{t} f(s, X(s))\, ds + \int_{t_0}^{t} B(s, X(s))\, dW_s \tag{8.8}$$

In general, analytic solutions to Eq. (8.8) exist only in the simplest cases and numerical integration must be used. Unfortunately, the same numerical integration procedures used for deterministic differential equations do not, in general, have analogs for their stochastic counterparts because of the different underlying calculuses [64]. Fortunately, the simplest method, the Euler method, does have an analog for stochastic differential equations. For SDEs, it is known as the Euler-Maruyama method [64].

The Euler-Maruyama method approximates Eq. (8.8) by using:

$$X_{t+1} = X_t + \Delta t f(t, X_t) + B\Delta W_n \tag{8.9}$$

where $\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}$, the change in the Brownian motion over the time interval $\Delta\tau = \tau_{n+1}$ - τ_n. Variable ΔW_n is a random variable with $\Delta W_n \sim \mathcal{N}(0, \Delta\tau)$. This enables the original SDE, Eq. 8.7, to be simulated. At each time step, sample a normal random variable with $\mu = 0$ and $\sigma^2 = \Delta\tau$. Then propagate the state forward using:

$$X_{t+1} = X_t + \Delta t f(t, X_t) + BZ \tag{8.10}$$

where Z is the sampled random variable. Eq. (8.10) is used as the basis for the numerical simulation of the dynamics in this chapter's shooting method trajectory optimization.

8.2.1 Dubins Car Case

Similar to Chapter 7, the Dubins car is used as a simple model for the methods in this chapter. Its equations of motion are repeated here for convenience:

$$\dot{x} = \cos(\theta) \tag{8.11}$$

$$\dot{y} = \sin(\theta) \tag{8.12}$$

$$\dot{\theta} = u_\theta \tag{8.13}$$

The stochastic differential equation for the Dubins car model is:

$$dX_t = \cos(\theta) + \sigma_x dW_t \tag{8.14}$$

$$dY_t = \sin(\theta) + \sigma_y dW_t \tag{8.15}$$

$$d\Theta_t = u_\theta + \sigma_\theta dW_t \tag{8.16}$$

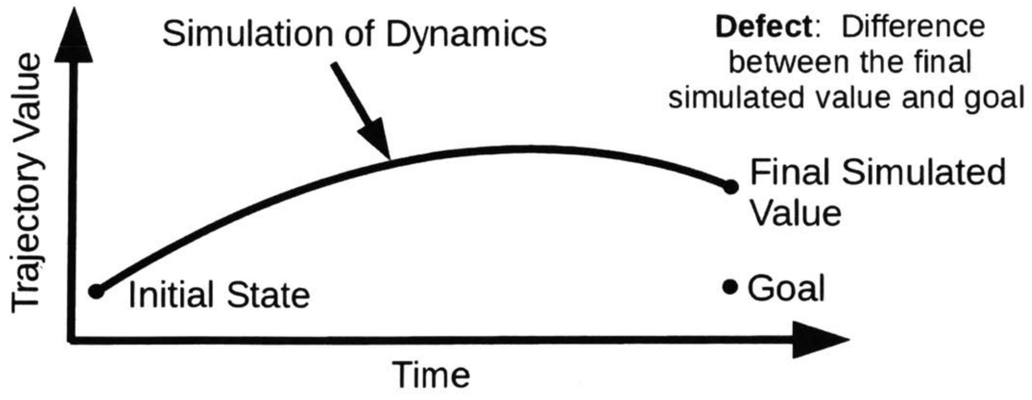


Figure 8-2: An illustration of the shooting method where a set of control inputs is simulated forward in time. The difference between the final simulated trajectory state and the final desired trajectory state is termed the defect.

The equations are discretized using Eq. (8.10):

$$X_{t+1} = X_t + h\cos(\theta) + \sigma_x\sqrt{h}Z_t \tag{8.17}$$

$$Y_{t+1} = Y_t + h\sin(\theta) + \sigma_y\sqrt{h}Z_t \tag{8.18}$$

$$\Theta_{t+1} = \Theta_t + u_\theta + \sigma_\theta\sqrt{h}Z_t \tag{8.19}$$

8.3 Trajectory Optimization via Shooting

A large approximation made in Chapter 7 was that all stochasticity was additive Gaussian noise. This was combined with a collocation trajectory optimization by adding Gaussian samples to the decision variables. This chapter focuses on creating a more accurate stochastic model for motion planning under uncertainty. Before this, the shooting method is introduced. This lays the foundation for introducing the Shooting Method Monte Carlo algorithm later in the chapter.

In contrast to direct collocation, shooting methods involve simulating a given choice of control inputs and iteratively correcting the resulting trajectory until it converges to a desired goal. This idea is depicted in Fig. 8.3. A multiple shooting method has multiple simulations and defects along a single trajectory. Each defect measures the amount that a trajectory segment mismatches the following segment.

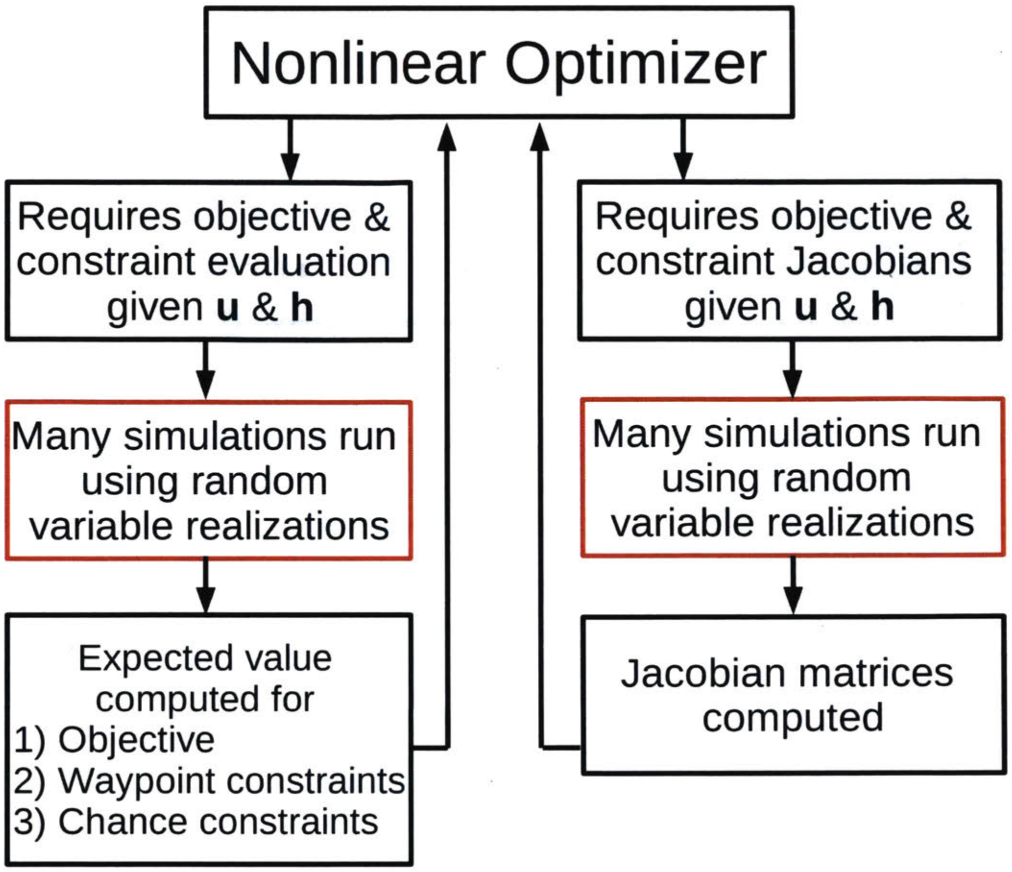

Nonlinear Optimizer
Requires objective & constraint evaluation given u & h
Many simulations run using random variable realizations
Expected value computed for
1) Objective
2) Waypoint constraints
3) Chance constraints
Requires objective & constraint Jacobians given u & h
Many simulations run using random variable realizations
Jacobian matrices computed

Figure 8-3: An overview of the Shooting Method Monte Carlo algorithm and its integration with an off-the-shelf nonlinear optimization library such as SNOPT. The approach is based on the shooting method described in Section 8.3 with the chief addition of simulating multiple trajectories, each a function of different realizations of the random variables. Summing over these trajectories approximates the expected value of the constraints and objective. The boxes in red may be parallelized.

In general, Jacobian matrices for shooting methods are more complex to generate because the state at each step t relies on control inputs at time steps 0 to $t-1$.

8.4 Shooting Method Monte Carlo

Shooting Method Monte Carlo (SMMC) is an approach to trajectory optimization designed to enable motion planning under uncertainty for nonlinear and therefore non-Gaussian dynamical systems. The insight to SMMC is that the shooting method for trajectory optimization is very similar to a single scenario of a Monte Carlo simulation. If a large number of simulations are run at each iteration of the trajectory optimization, then the probability of failure can be approximated. Simulating trajectories allows additional stochastic quantities to be computed as well, such as the expected trajectory cost.

SMMC proceeds as follows. The optimal control problem is transcribed into a mathematical program. Unlike the collocation method, the dynamical equations of motion are not enforced by explicit constraints but by forward simulating the dynamics. Additionally, the decision variables are not the states, control inputs $\mathbf{u}$, and time step vector $\mathbf{h}$ but only the control inputs and time step vector. An initial set of control inputs is used as guesses. Second, a set of samples is drawn from the additive distribution. The number of samples is SDT where S is the total number of scenarios, D is the number of dynamics states, and T is the planning horizon. At each iteration of the optimizer, the nonlinear optimization subroutine assigns values to the decision variables, $\mathbf{u}$ and $\mathbf{h}$, and asks for an evaluation of the objective and constraint functions or the Jacobian matrix of the objective and constraints. Given these decision variable assignments, the agent's trajectory is simulated across S scenarios by integrating the equations of motion. Each scenario requires using DT realizations of the random variables. By summing the results of the simulations, expected values are estimated for each requested function or gradient. The expected values are returned to the nonlinear optimization routine, which updates the decision variables. The process of updating the decision variables and simulating the trajectories continues until the nonlinear optimizer reaches a convergence criteria.

Because the trajectory is stochastic, it is impossible to guarantee that each of the constraints are satisfied. Therefore, the result of the simulations is used to approximated the expected value of each constraint. This is done by summing each constraint evaluation over all the scenarios. A similar strategy is used to compute the objective. Rather than computing a deterministic quantity for the objective, an expected cost is computed using the results of the simulations.

An example of the objective, constraint, and Jacobian computations for the Dubins car is provided in the following sections. An overview of the algorithm is shown in Fig. 8.4.

8.5 Formulation of the SMMC Program

An example formulation for the Dubins car use case is provided in the following sections.

A note on notation

In the following, F and G represent the objective and constraint functions respectively. Variable G_n represents the n^{th} constraint, with n total constraints. Time steps are indexed by k with a planning horizon of length K. For carrying out the Monte Carlo simulation, it is assumed there are a total of S scenarios with lowercase s indexing each scenario. The state vector is represented by $\mathbf{x}$, with a total of I states. States are indexed $x_{i,k,s}$ to represent the i^{th} state at the k^{th} time step for the s^{th} scenario. Similarly, $\mathbf{u}$ represents the control vector. Variable $u_{j,k}$ represents the j^{th} control input at time step k. Note that there is no scenario associated with the control input because the algorithm outputs a single deterministic controller; it is independent of the scenario. Variable $\mathbf{u}_{0:k}$ is used to denote all control inputs up to and including those at time step k. To avoid clutter, certain state and control indices are dropped when they are clear from context or not relevant to the equation.

Form of the Jacobian computation

As mentioned above, the Jacobian is more complex to compute than in the collocation case because of the need to account for the effect of control inputs from time t_0 to t_k on state $x_{k,s}$. The Jacobian requires computing the derivative of the objective and constraint functions with respect to the control inputs and time steps:

$$J(\mathbf{u},\mathbf{h}) = \begin{bmatrix} \frac{\partial F}{\partial u_{0,0}} & \frac{\partial F}{\partial u_{1,0}} & \cdots & \frac{\partial F}{\partial h_K} \\ \frac{\partial G_0}{\partial u_{0,0}} & \frac{\partial G_0}{\partial u_{1,0}} & \cdots & \frac{\partial G_0}{\partial h_K} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial G_N}{\partial u_{0,0}} & \frac{\partial G_N}{\partial u_{1,0}} & \cdots & \frac{\partial G_N}{\partial h_K} \end{bmatrix} \tag{8.20}$$

where F is the objective and G_n is the n^{th} constraint. Because the states $\mathbf{x}$ are no longer decision variables but are determined through simulation, they are functions of $\mathbf{u}$ and $\mathbf{h}$, i.e. $\mathbf{x}_{k,s}(\mathbf{u}_{0:k-1}, \mathbf{h}_{0:k-1})$. This means the value of constraint n at time step k, $G_{n,k}$, is possibly dependent on control inputs at all prior time steps. Fortunately, the Jacobian computation may be simplified by propagating certain gradients forward in time. Suppose we want to compute $\frac{\partial G_{n,k'}}{\partial u_{j,k}}$, the change in constraint $G_{n,k'}$ due to a change in control input $u_{j,k}$ where $k' > k$. The gradient $\frac{\partial G_{n,k'}}{\partial u_{j,k}}$ may be simplified using the chain rule of differentiation as:

$$\frac{\partial G_{n,k'}}{\partial u_{j,k}} = \sum_s^S \sum_i^I \frac{\partial G_{n,k'}}{\partial x_{i,k',s}} \frac{\partial x_{i,k',s}}{\partial u_{j,k}} + \sum_j^J \frac{\partial G_{n,k'}}{\partial u_{j,k'-1}} \tag{8.21}$$

The first term in the first sum on the right-hand side, $\frac{\partial G_{n,k'}}{\partial x_{i,k',s}}$,denotes the change in the constraint due to the change in the state at k' for scenario s. Its precise form depends on the constraint $G_{n,k'}$ and is covered in the following sections. The second term in the first sum represents the change in the state value $\partial x_{i,k',s}$ due to the change in control input $u_{j,k}$ where $k' > k$. In the case $k' - 1 = k$, $x_{i,k',s}$ may be an explicit function of $u_{j,k}$ and the derivative is straightforward to compute using the dynamics. In the case $k' - 1 > k$, $x_{i,k',s}$ is not an explicit function of $u_{j,k}$, it is necessary to compute this using the chain rule. Namely,

$$\frac{\partial x_{i',k',s}}{\partial u_{j,k}} = \sum_i^I \frac{\partial x_{i',k',s}}{\partial x_{i,k'-1,s}} \frac{\partial x_{i,k'-1,s}}{\partial u_{j,k}} \quad , k' - 1 > k \tag{8.22}$$

Note the recursive nature of Eq. (8.22). The RHS computes $\frac{\partial x_{i',k',s}}{\partial u_{j,k}}$ in terms of $x_{i,k'-1,s}$ while $\frac{\partial x_{i',k',s}}{\partial x_{i,k'-1,s}}$ can be computed from the dynamics. This suggests an approach for computing the gradients over the entire time horizon by maintaining terms $\frac{\partial x_{i',k',s}}{\partial u_{j,k}}$.

Algorithm 9: Method for computing the Jacobian matrix $\mathbf{J}$, Eq. (8.20), by propagating the gradients forward. $\mathbf{K}$ is a list containing all transition gradients, $\frac{\partial x_{i,k}}{\partial u_{j,k'}}$ for $k > k'$

```
Input: A set of I dynamics equations describing the
       evolution of the system state.
Output: The Jacobian matrix, J
1  T ← ∅
2  for s ← 1 to S do
3      for k ← 1 to K do
4          for $x_i \in \mathbf{x}_s$, $u_j \in \mathbf{u}$ do
5              Compute $\frac{\partial x_{i,k,s}}{\partial u_{j,k-1}}$ using Eq.(8.22) and store in K
6          end
7          for $k' \leftarrow 1$ to $k$ do
8              for $n_k \leftarrow 1$ to $N_k$ do
9                  $\frac{\partial G_{n_k,k}}{\partial u_{j,k'}} \leftarrow \frac{\partial G_{n_k,k}}{\partial u_{j,k'}} + \sum_i^I \frac{\partial G_{n,k}}{\partial x_{i,k-1,s}} \frac{\partial x_{i,k-1,s}}{\partial u_{j,k'}}$
10             end
11         end
12     end
13 end
```

Specifically, the dynamics equations are used to compute the terms:

$$\frac{\partial x_{i,1,s}}{\partial u_{j,0}} \ \forall \ i, j, s \tag{8.23}$$

These terms are used to compute $\frac{\partial x_{i,2,s}}{\partial u_{j,0}}$ using (8.22). This approach can be used to compute any $\frac{\partial x_{i,k',s}}{\partial u_{j,k}}$ for $k' > k$. Computation of the objective Jacobian proceeds similarly. Rather than compute the Jacobian for each of N constraints, there is only one objective:

$$\frac{\partial F}{\partial u_{j,k}} = \sum_s^S \sum_i^I \frac{\partial F}{\partial x_{i,k',s}} \frac{\partial x_{i,k',s}}{\partial u_{j,k}} + \sum_j^J \frac{\partial F}{\partial u_{j,k'-1}} \tag{8.24}$$

The method of computing gradients and propagating them forward is shown in Alg. 9.

8.5.1 Objective

Similar to the collocation case in Chapter 7, the Euclidean distance may be used as an objective function. In terms of the 2D position states, this is:

$$C(\mathbf{x}) = \sum_{k}^{K-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2} \tag{8.25}$$

For the shooting method, the dependence on the control inputs may be made explicit, because each (x_k, y_k) state is computed as the result of a simulation dependent on $\mathbf{u}_{0:k-1}$:

$$C(\mathbf{u}) = \sum_{k}^{K-1} \sqrt{(x_{k+1}(\mathbf{u}_{0:k}) - x_k(\mathbf{u}_{0:k-1}))^2 + (y_{k+1}(\mathbf{u}_{0:k}) - y_k(\mathbf{u}_{0:k-1}))^2} \tag{8.26}$$

An advantage of SMMC is that the expected is straightforward to compute after performing the trajectory simulations. This is in contrast to the collocation method of Chapter 7, which relied on the deterministic cost. The expected cost is approximated by summing over all scenarios:

$$\mathbb{E}[C(\mathbf{u})] = \sum_{s}^{S} \sum_{k}^{K-1} \sqrt{(x_{k+1,s} - x_{k,s})^2 + (y_{k+1,s} - y_{k,s})^2} \tag{8.27}$$

Objective Jacobian

Computing the Jacobian matrix requires the approach outlined in subsection 8.5. What remains is to compute $\frac{\partial F}{\partial x_{i,k'}}$, the derivative of the objective with respect to the states. Given the objective in (8.27), there are four derivatives that must be computed: the derivative of the objective with respect to x_{k+1}, x_k, y_{k+1}, and y_k:

$$\frac{\partial F}{\partial x_{i,k+1,s}} = \frac{x_{i,k+1,s}}{\sqrt{(x_{k+1,s} - x_{k,s})^2 + (y_{k+1,s} - y_{k,s})^2}} \tag{8.28}$$

$$\frac{\partial F}{\partial x_{i,k,s}} = \frac{-x_{i,k,s}}{\sqrt{(x_{k+1,s} - x_{k,s})^2 + (y_{k+1,s} - y_{k,s})^2}} \tag{8.29}$$

$$\frac{\partial F}{\partial y_{i,k+1,s}} = \frac{y_{i,k+1,s}}{\sqrt{(x_{k+1,s} - x_{k,s})^2 + (y_{k+1,s} - y_{k,s})^2}} \tag{8.30}$$

$$\frac{\partial F}{\partial y_{i,k,s}} = \frac{-y_{i,k,s}}{\sqrt{(x_{k+1,s} - x_{k,s})^2 + (y_{k+1,s} - y_{k,s})^2}} \tag{8.31}$$

These equations are used along with Eq. (8.24) to compute the objective gradients.

8.5.2 Waypoints

The SMMC implementation relies on FOX's waypoint formulation. Similar to the method in Chapter 7, the chance constraint is computed using sampling and the only necessary waypoint coordinate is $\hat{w}$. Rather than constrain the position state to lie on FOX's constraint surface, the expected value of the position state is required to lie on the constraint surface, $\mathbb{E}[w_n] = 0$. This is approximated by summing over the scenarios,

$$\mathbb{E}[w_n] \approx \sum_{s}^{S} (\boldsymbol{\omega}_{n,s} - \vec{o}_n) \cdot \vec{w}_n = 0 \tag{8.32}$$

where $\boldsymbol{\omega}_{n,s} = (x, y)_{n,s}$ is the position state for scenario s at waypoint n. The gradients are relatively straightforward to compute:

$$\frac{\partial w_n}{\partial x_{i,k=n,s}} = \vec{w}_{n,x} \tag{8.33}$$

$$\frac{\partial w_n}{\partial y_{i,k=n,s}} = \vec{w}_{n,y} \tag{8.34}$$

These equations can be used in Eq. (8.21) to compute the gradients.

8.5.3 Chance Constraint

The chance constraints also make use of Eq. (8.21) where the $\frac{\partial G_{n,t'}}{\partial x_{i,t',s}}$ is determined by workspace states. These terms are defined in Chapter 7.

8.5.4 Transition Gradients

The transition gradients, i.e. gradients representing $\frac{\partial x_{t+1}}{\partial u_t}$, require use of the equations of motion for computation. As described above, the Dubins car dynamics using Euler-Maruyama integration take the form:

$$x_{t+1} = x_t + h_t \cos(\theta_t) + \sigma\sqrt{h_t}Z \tag{8.35}$$

$$y_{t+1} = y_t + h_t \sin(\theta_t) + \sigma\sqrt{h_t}Z \tag{8.36}$$

$$\theta_{t+1} = \theta_t + u_\theta \tag{8.37}$$

The gradients are straightforward to compute:

$$\frac{\partial x_{t+1}}{\partial h_t} = \cos(\theta_t) + \frac{\sigma Z}{2\sqrt{h_t}} \tag{8.38}$$

$$\frac{\partial y_{t+1}}{\partial h_t} = \sin(\theta_t) + \frac{\sigma Z}{2\sqrt{h_t}} \tag{8.39}$$

$$\frac{\partial \theta_{t+1}}{\partial u_\theta} = 1 \tag{8.40}$$

8.5.5 Parallelization on GPUs

Shooting Method Monte Carlo also takes advantage of the massive GPU parallelization discussed in Chapter 7. Rather than simply parallelize the collision checks, as was possible in the collocation model, it's possible to parallelize the simulation of each scenario. This means that the parallelization can help evaluation of the chance constraint, waypoint constraints, dynamics, and objective. The same holds for evaluation of the Jacobian. This is depicted in Fig. 8.4.

8.6 Importance sampling

One of the principle drawbacks of Monte Carlo methods is their slow convergence. From the central limit theorem, Monte Carlo methods in general converge at a rate of $\sqrt{N}$, where N is the number of samples. This means to half the variance, four times as many samples are needed. A further difficulty is that, when sampling from complex

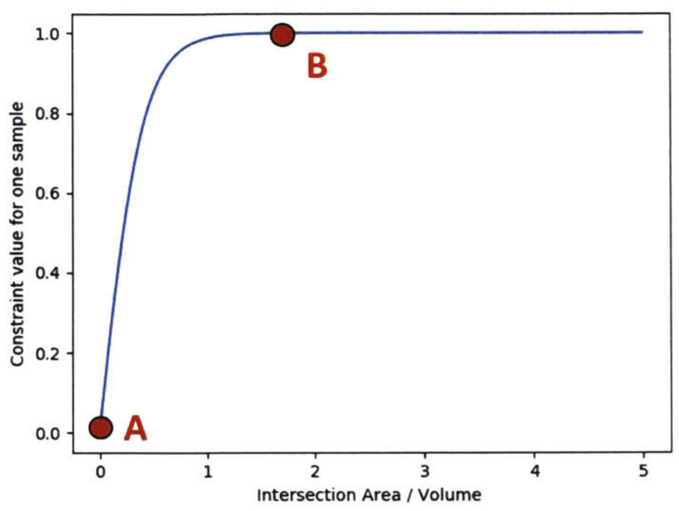


Figure 8-4: Illustration of two collision checks that generate a zero gradient and could cause the nonlinear optimizer to fail.

distributions, only a small number of samples may be relevant to the distribution's quantity of interest [81]. This tends to be the case in Shooting Method Monte Carlo; only a small number of samples are relevant to the optimization.

The optimizer evaluates the chance constraint and its gradients to compute an optimized policy. As described in this chapter and in Chapter 7, a large number of scenarios are used to compute the trajectory's risk. The intersecting region between the agent and obstacles is input into a sigmoid function to generate an approximation of the collision indicator function. The sigmoid allows the collision to generate a non-zero gradient over a wider range; the non-zero gradient helps the optimizer determine decision variables that satisfy the constraint. A sigmoid with a greater slope will better approximate the collision function; however, it makes it less likely that any scenario has a non-zero collision check gradient.

This problem is illustrated in Fig. 8-4. This figure illustrates the constraint value as a function of the intersecting region for a scenario. Point A indicates that the scenario is not in collision. All scenarios with an intersecting region of size zero at at point A and have zero gradients. This is OK because these paths do not affect feasibility with respect to the chance constraints. Point B represents a scenario that is in collision. At point B, the region in collision is large, the sigmoid has value 1, and

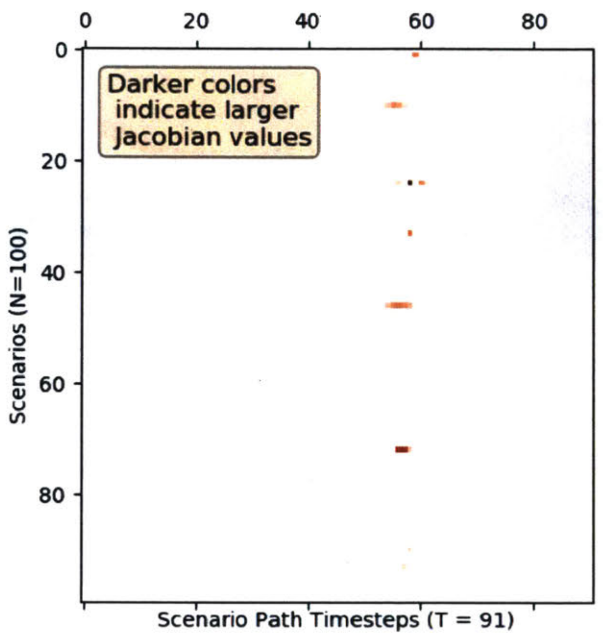


Figure 8-5: Plot illustrating the sparse nature of the non-zero collision gradients.

the gradient is zero. If too many scenarios are at similar points as B, the algorithm tends not to converge because it is unable to remove these scenarios from collision. As the sigmoid slope becomes steeper, it becomes more likely that more scenarios lie at points A and B and very few have non-zero slopes. This means it's likely that very few scenarios (i.e. a very small amount of the computational effort) are contributing to the algorithm's convergence. If no scenarios have non-zero gradients, the algorithm tends not to converge.

A similar problem can occur in time. Over a time horizon, it's likely that only a small number of time steps are in collision and contribute to solving the problem. It makes sense to try and concentrate efforts around these scenarios.

The sparsity of the Jacobian is shown for a test case in Fig. 8-5. The darker red indicates larger values of the Jacobian; the majority of the figure is light red where the gradients are zero. Of the 100 scenarios, only a handful have a nonzero Jacobian; the same is true with respect to time steps.

In order to make the Jacobian less sparse and encourage convergence of the algorithm, I propose a re-sampling strategy. The re-sampling proceeds as follows by

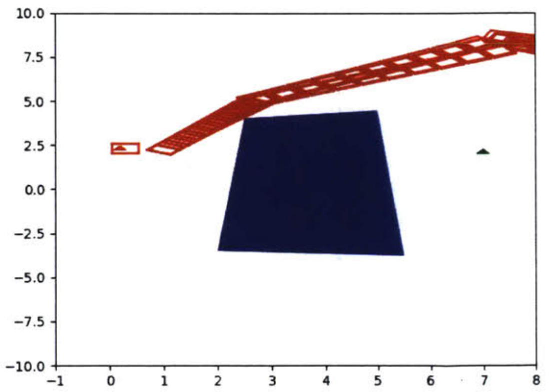


Figure 8-6: Simulations of trajectories using an initial guess for the control inputs **u**.

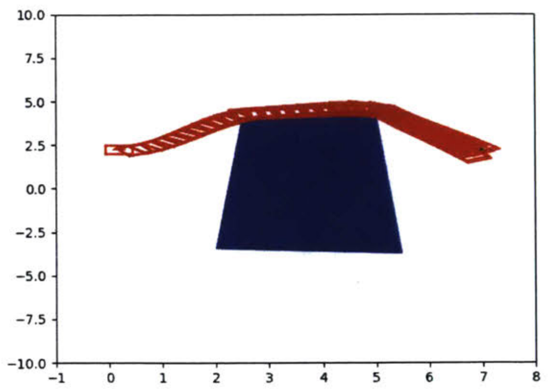


Figure 8-7: Simulations of trajectories using the optimized control inputs.

re-solving the problem and varying the sigmoid function. For convenience, the sigmoid function is repeated:

$$G_{collision} = \frac{1}{1 + \exp\left(-\alpha|\mathcal{W}|\right)} \tag{8.41}$$

where $|\mathcal{W}|$ is the size of the intersecting region and α is a scaling constant. Increasing α makes the sigmoid steeper, and the derivative non-zero over a narrower range, but better approximates the collision check indicator function. The algorithm begins by computing a trajectory using a low α. This makes it more likely that certain scenarios will lie in the region of the sigmoid with non-zero gradient and the nonlinear optimizer will be able to find an initial solution to the problem. Given this initial solution, a new set of samples is generated using those scenarios with a non-zero collision gradient in the first solution. Given these new samples, the problem is re-solved using a larger α that more accurately reflects the true collision boundary. This process repeats until a threshold is reached on α.

8.7 Results

The algorithm was tested with the Dubins car model. An example set of simulations is shown in Figs. 8-6 and 8-7. Further benchmarks are shown in Figs. 8-8 - 8-9 and Table 8.1.

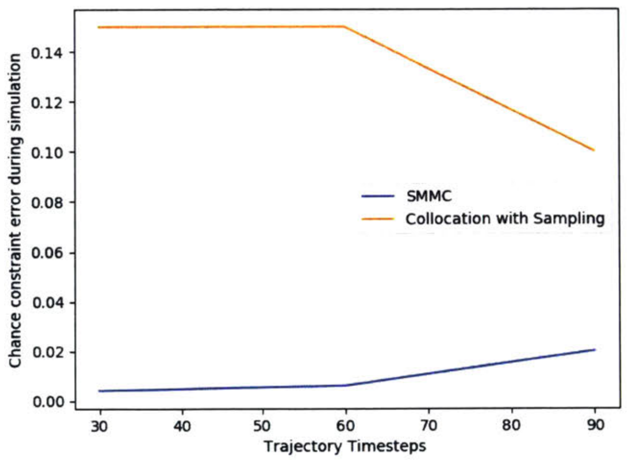


Figure 8-8: Comparison of the ability of Shooting Method Monte Carlo to approximate the chance constraint in simulation versus a sampling-based collocation method.

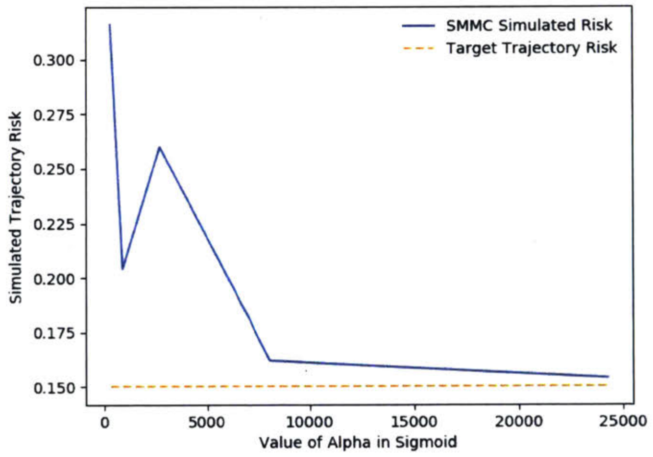


Figure 8-9: Plot depicting SMMC's ability to approximate the chance constraint in simulation as a function of the sigmoid's α parameter.

Number of Scenarios	Serial (secs)	Parallel (secs)	Hot Start
100	28.2	16.1	No
137	6.4	2.3	Yes
149	9.3	3.1	Yes
3732	182.2	24.3	Yes

Table 8.1: Table illustrating the speed improvement enabled by parallelization of SMMC on a GPU.

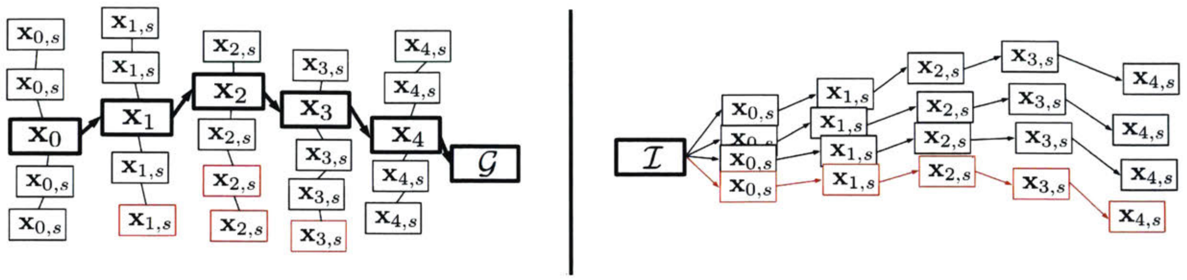


Figure 8-10: Differences in simulation between Chapter 7 and Chapter 8. The pane on the left illustrates the collocation approximation to the chance constraint; samples are independent between time steps. The right pane illustrates the model assumed by SMMC. Red boxes indicate samples / scenarios that may be in collision.

The simulation in Fig. 8-6 shows simulations of Dubins car trajectories given an initial guess for the vector $\mathbf{u}$. As mentioned previously, the initial trajectory should be feasible and is, in general, non-trivial to determine. Fig. 8-7 shows the resulting output trajectory, which respects the chance constraint.

The SMMC approach was benchmarked against the sampling-based approach discussed in the previous chapter for their abilities to produce trajectories that match the chance constraint in simulation. The collocation trajectory optimization was performed as described in Chapter 7; agent dynamics are modeled via collocation constraints. The collocation chance constraint is modeled via Eq. (7.28) and computed by sampling from the agent's configuration state at various collocation knots. The trajectories generated via the collocation method and shooting method were then simulated, the simulation being slightly different from that described in Chapter 7 to benchmark the sampling-based method against CDF-based chance constraints. The simulation described in Chapter 7 assumed the agent followed its nominal trajectory as output by the trajectory optimizer; samples were drawn from this nominal trajectory and the ratio of samples in collision used to approximate the simulated trajectory risk. This idea is shown in the left hand pane of Fig 8-10. The main simplification assumed by this simulation (and the methods discussed in Chapter 7) is that the samples are independent between time steps. This chapter uses a simulation that does not place the agent on a nominal trajectory. Specifically, this chapter uses a scenario-based simulation. If the configuration of the agent at time step t is $\mathcal{C}_{t,s}$

where s denotes the current scenario, the agent's configuration at step $t+1$ is given by:

$$\mathcal{C}_{t+1} = \mathcal{C}_t + \sqrt{h_t}\Sigma \boldsymbol{Z} \tag{8.42}$$

where Σ is a covariance matrix and $\boldsymbol{Z}$ is a vector of standard normal samples. This is a more accurate simulation method because it captures the dependence between time steps. This is depicted in the right pane of Fig 8-10.

Fig. 8-8 shows the result of comparing simulations of the sampling-based collocation method with Shooting Method Monte Carlo. The sampling-based collocation method is unable to match the simulated trajectory risk because it does not model dependencies between time steps. There is error in assuming that the trajectory risk is modeled by drawing Gaussian samples from the nominal trajectory because this does not match the simulated state distribution. On the other hand, SMMC is able to produce trajectories that are close in simulation to the desired trajectory risk. In Fig. 8-8, the risk was set to 0.15; therefore, the collocation method often missed the obstacle entirely in simulation and produced trajectories with very little risk. SMMC was run using re-sampling stages as described in 8.6. Both collocation and SMMC tests used similar numbers of scenarios. SMMC exhibits slightly more erorr in the case of the trajectory with 90 time steps, likely due to the length of the trajectory and the increased number of scenarios necessary to approximate the chance constraint for longer scenarios.

Fig. 8-9 shows how increasing the value of α in Eq. (8.41) improves the accuracy of SMMC in respecting the chance constraint. As α increases, the sigmoid function approaches the indicator function that indicates whether a scenario is in collision or not.

Finally, Table 8.1 provides examples of how parallelization speeds up SMMC. The second column shows the time taken to run SMMC using only a CPU without parallelization. The third column shows the algorithm run on the same test cases but using parallelization to simulate trajectories and determine whether each is in collision. The final column, *Hot Start*, indicates whether the optimization was initialized with a solution from a previous optimization. The table shows that parallelization reduces the algorithm run time with the largest speed up coming when the number of scenarios is large. The makes sense; when there are a fewer number of scenarios to simulate, the time spent in the nonlinear optimizer dominates the overall run time. When there is a large number of scenarios, the time spent simulating each scenario dominates the run time.

Chapter 9

Resource Allocation Under Uncertainty

The final three chapters of this thesis examine planning under uncertainty for systems with limited computational resources. As mentioned earlier in the thesis, planning under uncertainty is often intractable because of the need to consider large numbers of scenarios. Because it can be computationally-intensive, planning under uncertainty is best suited for high cost, cutting edge hardware. However, considering stochasticity is necessary for certain low cost systems as well. Electricity scheduling in the developing world provides an example of this.

The past decade has seen vast changes in the way electricity is generated, delivered, and consumed. The price of renewable energy has decreased to the point of making it competitive with conventional energy resources in many markets without subsidies [50]. Growing interest in energy efficiency and electric vehicles could mean dramatically reduced demand for petroleum in transportation. In addition to these trends, artificial intelligence and autonomous agents play an increasing role in energy. On the supply side, the increasing penetration of renewable energy presents not only an opportunity to reduce carbon emissions but also a threat to destabilize the grid due to their intermittent nature [67]. On the demand side, there has been a proliferation of smart devices that are able to act as electricity "pro-sumers," i.e. changing their power consumption based on the availability or price of electricity [114].

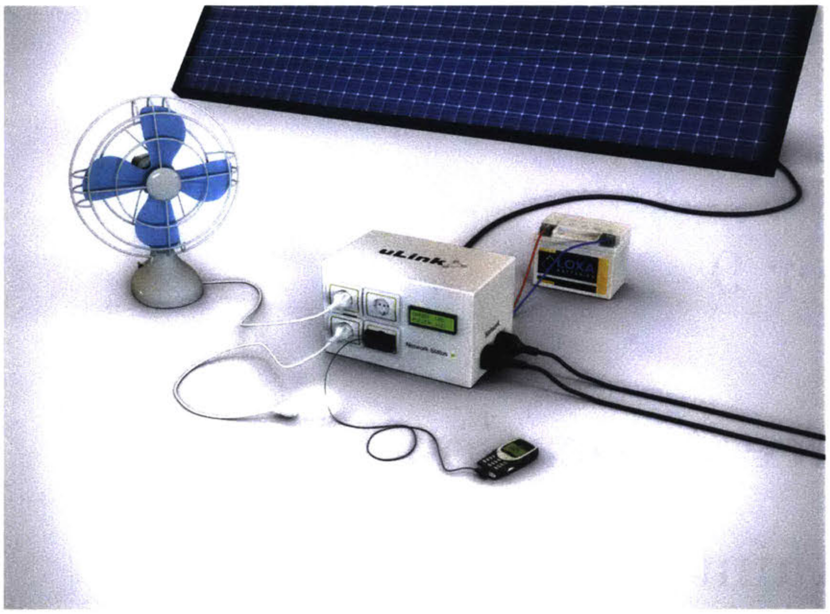


Figure 9-1: 3D drawing of the uLink device. The device could be connected to a variety of DC loads and electricity sources. A battery is crucial for dealing with unpredictable solar generation.

In spite of these rapid advancements, one out of every seven people on Earth lack reliable electricity. Recently, solar-powered microgrids have been proposed as a solution to this problem. Microgrids are electricity grids that can operate using electricity from the national grid, or, more frequently, exist as islanded grids that generate their own power. Microgrids have the potential to electrify parts of the developing world because they require less planning than extending the grid and can provide a lower cost of electricity. However, achieving this goal requires solving a complex problem in resource allocation under uncertainty.

9.1 uLink Project

This work was part of the uLink project, a technology enabling ad hoc microgrids. The uLink was a shoebox-sized device to which users could connect DC loads such as cell phones and lights. On the generation side, users could also connect solar panels and batteries to the device. Importantly, the devices could network together such that ad hoc microgrids could form. This was important because it meant that users

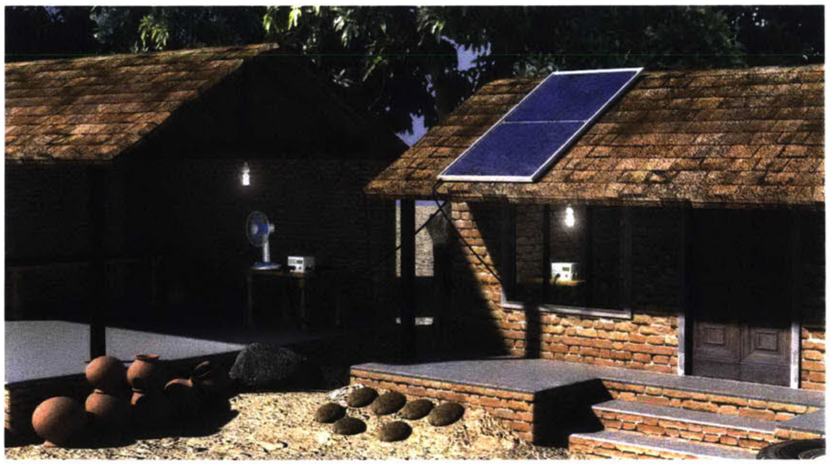


Figure 9-2: uLink devices could be connected together to from an ad hoc microgrid. Certain users would have generating and storage assets while others would only have loads. The key insight to the uLink technology was that users were able to share energy; users with generating assets could "sell" electricity to users without generation. To do this autonomously, resource allocation algorithms were necessary that could run on the ultra low cost hardware.

with generating assets could share or sell electricity to other users. One of the biggest problems extending the electricity grid and traditional microgrids is that they require very long and expensive planning procedures. Grid operators need to be ensured the demand will roughly meet the supply and the grid makes economic sense. For uLink, users would be incentivized to connect to the grid because they could sell electricity. Resource allocation algorithms are crucial in this environment because they enable the grid to operate autonomously while ensuring users had certain qualities of service. Images of the uLink device and its networking capability are shown in Images 9-1 and 9-2.

The uLink device sought to have an incredibly low price point (the device itself projected to cost about $15 and running a $2 microcontroller). These circumstances presented a challenging for planning under uncertainty in an extremely resource-constrained environment. The microcontroller itself had less than 1 MB of memory. The next two chapters detail resource-allocation algorithms developed to solve the problem of planning under uncertainty in a computationally-constrained environment.

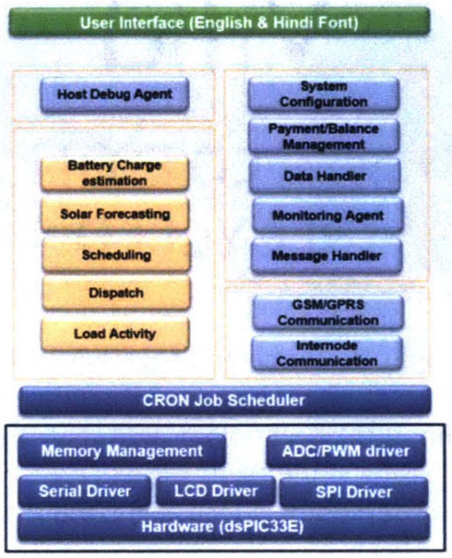


Figure 9-3: uLink operating system and software architecture.

Figure 9-4: The uLink device operating during a field trial in Jharkhand, India.

9.2 Resource Allocation Under Uncertainty

Similar to motion planning under uncertainty, even in the deterministic case, resource allocation is a very hard problem. Because the 0-1 knapsack problem is an instance of resource allocation, even the simplest case is an NP-Hard problem. The deterministic problem consists of deciding whether to schedule a set of $\mathcal{A}$ activities over some time horizon T given some resource availability defined over the time horizon $R(t)$. In this formulation, the problem is very similar to 2D bin packing. If resource availability is on the x axis and time on the y axis, activities can be thought of as having two dimensions: a resource requirement and a time requirement. The activities must be "packed" into a two dimensional bin where the height measures the available resource at some time.

The deterministic energy resource allocation problem is more challenging than this baseline example of resource allocation because of batteries. In resource allocation on a microgrid, $\mathcal{A}$ represent some form of electrical activities that users would like to accomplish and the resource $R(t)$ is the amount of energy available on the grid. This energy may be generated from solar panels or diesel generators. Batteries mean that a certain amount of resource may be shifted (up to the battery's capacity). This is key to the microgrid being able to deliver critical nighttime loads, especially when only solar generation is available during the day. Unfortunately, the ability to shift energy availability makes scheduling more difficult.

Resource allocation under uncertainty is more difficult because resource demand and generation have probability densities associated with them. In the case of demand, users may want varying quantities of resources at various times of the day. For example, a user on a microgrid may want lighting at nighttime but do not have a precise idea of when or how much they want. On the generation side, uncertainty in the amount of insolation means the amount of energy generated during the day is stochastic. Uncertainty in supply and demand leads to a problem seen during the first part of this thesis with motion planning under certainty: intractability. An algorithm must compute a policy (a mapping from states to actions) given that the state of the system is uncertain. It's necessary to solve both the resource allocation problem and determine and efficient state representation under uncertainty.

9.3 Prior Work

A great amount of work has been pursued on grid management under uncertainty. One approach is to formulate the problem as a stochastic program and solve using a standard mixed-integer linear program solver [16], [66], [119]. The stochastic programming approach has also been used to control micro-grid operation [77]. Stochastic programs can deliver high quality solutions; however, they require solving very large MILPs, which is too computationally intensive for our low-cost environment. Additionally, most stochastic programs rely on centralized solutions; microgrid management is often better suited for multi-agent approaches since competing agents may not want to reveal their utility functions to centralized solvers.

The problem of smart grid power management has also seen extensive interest in the AI and multi-agent communities where the problem is often tackled from the perspective of game theory and mechanism design [94]. Multi-agent coordination is explored in [113], where an efficient Continuous Double Auction is deployed to manage competing user interests and transmission line constraints. Game theory has also been used for applications in the energy access space such as in [2] where an efficient negotiation protocol is created that allows users to trade electricity in developing world microgrids, which is similar to our application. In [107], a Stackelberg game is studied to create a consumer-centric energy trading mechanism and encourage user participation. A key theme of economic-based approaches is the use of prices to signal to agents when the electricity network is over or undersupplied. A novel scheduling algorithm is presented in [109] where a central utility provides pricing signals to distributed agents.

A market-based approach seeing recent interest is tatonnement. Tatonnement is a distributed optimization method that mirrors dual decomposition [19] and has been widely used for resource allocation [47]. In tatonnement, distributed agents use local utility functions to place a bid for a quantity of a good. A market then determines whether or not the bids represent a feasible allocation of the good. If demand is higher/lower than supply, the market increases/decreases the good's price and reports it to the nodes. Using the updated prices, the nodes recompute their new demand. The price is again raised or lowered and the process repeats until convergence. A recent application of tatonnement to grid scheduling is described in [51], which develops a fast-converging version of tatonnement through utilizing randomized scheduling. Much of the work using market and pricing mechanisms builds on game theory. One example in [94] uses an auction mechanism to optimize both user utility and grid utility. This and other market-based work relies on expert knowledge, such as users' exact objective functions or which loads are controllable and which non-controllable. In real life, this information is difficult and tedious to obtain.

Overall, many of the ideas in literature focus on computationally-intensive mixed integer linear program (MILP) solvers to determine a policy. While the market-based methods show promise because they can distribute computational resources, most focus on the deterministic case. It is towards these areas that the following two chapters focus: designing distributed and centralized approaches for resource allocation under uncertainty on extremely resource-limited hardware.

Chapter 10

Distributed Resource Allocation

10.1 A Market for Reliability

The Market for Reliability algorithm proceeds similar to tatonnement. A set of agents would like electricity at certain times with a guaranteed quality of service. Agents bid on the probability that their activity is dispatched by a central market under uncertain solar generation. The reliability auction continues until agents have converged, scheduling themselves in 1) time and 2) reliability. The bidding process is illustrated in Fig. 10-1. The following sections describe the algorithm in detail.

10.1.1 Problem Formulation

The market is formulated in terms of a set of agents, $\boldsymbol{N}$, which bid for power in terms of activities. Each agent has a vector of desired activities that it would like to perform. The activities are defined similar to [109], however with the addition of a *tier* field that indicates a quality of service preference. An activity is defined as a tuple of 4 elements:

$$A_{desired,i,k} = \{lb, ub, P(t), tier\} \tag{10.1}$$

where $A_{desired,i,k}$ represents the k^{th} desired activity of agent i, lb is a lower bound of the time i would like A to start, ub is an upper bound on the time, $P(t)$ is the activity's power profile as a function of time, and $tier$ represents a quality of service

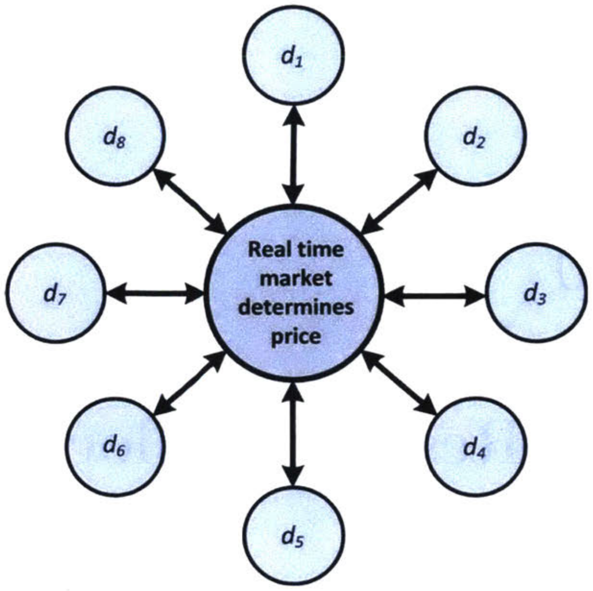


Figure 10-1: Graphical illustration of the algorithm; agents place bids for activities for their desired quality of service. A central market uses Monte Carlo simulation to determine whether the bids allow for a feasible resource allocation.

associated with the activity. The *tier* variable defines the utility function that an agent will use to bid for an activity's reliability. An activity representation is suitable for this problem because it is how developing world users typically view power.

10.1.2 Bidding on Reliability

To schedule activities, this chapter leverages work in [118] and uses a risk-based tatonnement algorithm. In traditional tatonnement the good is normally a physical good such as corn, oil, or electricity. In our formulation, the good is reliability, where reliability refers to whether or not the network will actually serve the activity. Specifically, an agent i would like to perform some activity A where A is defined by (10.1). We define the reliability of A as the probability that the network is able to serve the activity over a specified time horizon, $T = \{t_1, ..., t_N\}$. Qualitatively, because of uncertainty in generation, network configuration, and failures, no activity can be guaranteed that it will be served with probability one. Therefore, agents make bids directly on the probability that their activity will be served. The form of the bid is the 3-tuple, $bid_i = \{startTime, price, P(t)\}$ where $bid.startTime$ refers to when

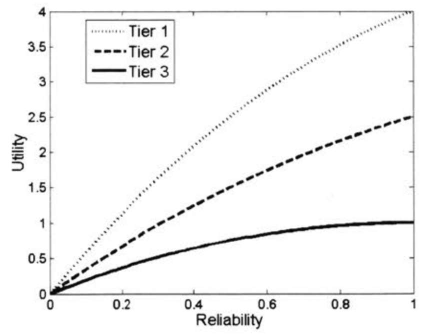


Figure 10-2: Utility functions representing three tiers of service.

agent i would like its activity to start, *bid.price* refers to the price the agent is willing to pay for reliability, and *bid.*$P(t)$ is a power profile as a function of time.

10.1.3 Local Utility Functions

To determine their desire for reliability, each activity has associated with it a local utility function. Similar to [118], the utility functions considered are quadratic functions linear in price:

$$U_i(t, r) = a_i r^2 + b_i r - p(t, r)r \tag{10.2}$$

In this equation, constants a_i and b_i represent an agent's desire for reliability, r represents the reliability in decimal form, and $p(t, r)$ is the price for reliability, which will be covered in the next section. Examples of utility functions are shown in Figure 10-2, where agents in a higher tier bid more for higher reliability.

For this paper, the utility functions selected for different tiers were arbitrary. In our microgrid implementation in India, the tier and corresponding utility function were assigned by the grid operator to the user, who pays a certain premium for a higher tier. One advantage of our approach is that grid operators can sell users utility functions that are easily translated into explicit chance constraints on the quality of service, i.e. the probability a user receives power is greater than some percent, $P(A) > x$.

10.2 Determining Bid Feasibility

After nodes submit bids, it must be determined whether they are feasible. That is, given a set of bids, $\mathbf{B}$, where $B_i = P(A_{demanded,i})$, it must be determined whether $P(A_{actual,i}) = P(A_{demanded,i})$. For example, if $P(A_{actual,1}) = 0.75$ and the scheduling horizon is one day, then the grid can serve Activity 1 on 75% of days. Two innovations are detailed in this section. First, I describe how the network can impose the notion of fairness through a probabilistic ordering on activities and second provide a simple Monte Carlo method for determining bid feasibility. This second innovation allows feasibility to be determined on low-cost microcontrollers with limited memory.

10.2.1 Creating Solar Generation Scenarios

To determine $P(A_{actual,i})$, the probability of serving A is first conditioned on a generation scenario. Define a generation scenario as a discretized realization of generation capacity for one scheduling horizon, $\mathbf{S_i} = [s_{i,1}, ..., s_{i,N}]$. Scenario $\mathbf{S_i}$ is created by sampling from the density function for the solar generation, deploying a standardized method, the Standard Average Approximation (SAA) Method described in [63]. The continuous distribution is approximated by a weighted sum of the scenarios:

$$\begin{aligned} P(A_{i,actual}) &= \int P(A_i|\mathbf{S} = \mathbf{s}) f(s_1, ..., s_N)\, d\mathbf{s} \\ &\approx \sum_i^K P(A_i|\mathbf{S} = \mathbf{s}) P(\mathbf{S} = \mathbf{s}) \end{aligned} \tag{10.3}$$

10.2.2 Ordering Activities Based on Price Paid

Given a generation scenario, $P(A_i|\mathbf{S} = \mathbf{s})$, must be determined. If the grid is under-supplied, it is necessary to decide on which activities are given priority. One method to determine priority is to use a deterministic ordering, where activities with higher priority are always served before activities with lower priority as described in [57]. This has the drawback that lower-tiered users may never receive power if the network is always undersupplied. Instead, we impose a probabilistic ordering where activities are more likely to receive power based on the price they paid for reliability.

A simple two activity example is as follows. Given a network with 10 kWh of energy available and two activities, both of which require 10 kWh, the network must decide which activity to serve. Node 1 has bid 10 units for its activity, A_1, to be served while Node 2 has bid 5 units for its activity, A_2, to be served. Given these bids, A_1 will be selected with probability:

$$P(A_1) = \frac{10}{15} = 0.66 \tag{10.4}$$

while the probability of serving A_2 is:

$$P(A_2) = \frac{5}{15} = 0.33 \tag{10.5}$$

For two activities this ordering analytically is straightforward to compute. However, the number of terms required to compute the probabilistic ordering is combinatorial in the number of activities because one needs to consider every possible ordering of activities. For example, the probability of serving Activity 1 under a certain generation scenario in a three activity case is:

$$\begin{aligned} P\left(A_{1,t}|S=s\right) =& P\left(A_{1,t}|S=s, A_{1,2,3}\right) P\left(A_{1,2,3}|S=s\right) + \\ & P\left(A_{1,t}|S=s, A_{1,3,2}\right) P\left(A_{1,3,2}|S=s\right) + \\ & P\left(A_{1,t}|S=s, A_{2,1,3}\right) P\left(A_{2,1,3}|S=s\right) + \\ & P\left(A_{1,t}|S=s, A_{2,3,1}\right) P\left(A_{2,3,1}|S=s\right) + \\ & P\left(A_{1,t}|S=s, A_{3,1,2}\right) P\left(A_{3,1,2}|S=s\right) + \\ & P\left(A_{1,t}|S=s, A_{3,2,1}\right) P\left(A_{3,2,1}|S=s\right) \end{aligned} \tag{10.6}$$

Where $P\left(A_{1,2,3}|G_t = g_t\right)$ is the probability of serving Activities 1, 2, and 3 in that order, given that the generation is g_t and $P\left(A_{1,t}|G_t = g_t, A_{1,2,3}\right)$ is the probability that A_1 receives power given the ordering. This is a binary value dependent on whether or not there is enough energy available to serve A_1.

Because of the combinatorial nature of Eq. 10.6, $P\left(A_{1,t}|G_t = g_t\right)$ is determined through a Monte Carlo method by simulating a given scenario a large enough number of times and then summing over the times each activity is served:

$$P\left(A_{i,actual}|G = g\right) \approx \frac{1}{K}\sum_{j=1}^{K} \mathbf{1}_{A_{i,served}} \tag{10.7}$$

During each simulation, the probability of choosing one activity i over others is the bidding price of one activity over the sum of the remaining activities that have not yet been selected in the simulation:

$$P\left(\text{Selecting i}\right) = \frac{p_i}{\sum_{remaining} p_k} \tag{10.8}$$

Where uppercase P refers to probabilities and lowercase p refers to bidding prices. For a given scenario, activities are selected and power allocated until further allocation is impossible (the implementation is described in Algorithm 11). Eq. 10.3 is then used to determine the overall probability of serving A_i.

The above section describes how a central market determines the actual probability that the network can serve an activity, given its price paid. The next section describes how agents create these prices.

10.3 Bidding Strategy

Agents submit bids of the form described in section 10.1.2. Associated with this bid is a price, which determines how likely an agent is to receive electricity relative to other agents, described in subsection 10.2.2. This section describes the form that these prices take and how agents decide the time of their bid.

In traditional tatonnement, there is typically one price per unit of good; for example, in a market where oil costs \$100 per barrel, an agent can bid \$500 for 5 barrels. In this case, because it's necessary to schedule activities in both time and reliability, a more robust pricing scheme is needed. Care must be taken that it leads to convergence.

The following pricing scheme and bidding strategy are proposed. This satisfies the requirements of allowing a distributed network of agents to converge to a feasible schedule in both time and reliability, is lightweight, and is guaranteed to converge. It takes inspiration from multiagent demand response literature where agents use pricing signals to determine when a grid is oversupplied or undersupplied [112].

10.3.1 Form of the Price

Each activity is assigned a pricing vector, which will only be used by that activity. The pricing vector has a price for each time step during the activity's duration. For example, given an activity described by (10.1) with *lb* and *ub* on the time it would like to begin, a unique pricing vector is assigned:

$$\boldsymbol{p_i} = [p_{i,t=lb}, p_{i,t=lb+1}, \ldots, p_{i,t=ub-1}, p_{i,t=ub}] \tag{10.9}$$

where $p_{i,t}$ is the price bid by activity i for reliability at time t *if* it decides to bid for that time. The advantage of using a vector to represent prices is to recognize that reliability is more expensive at certain times due to excess demand on the network. As agents compete for reliability, prices fluctuate at each time step throughout the auction.

10.3.2 Price Updates

Agents must schedule themselves using only pricing signals provided from a central market. After an agent i submits a bid, the market responds with the agent's actual reliability, $r_{act,i,t}$, at that time t for the bid price, $p_{i,k,t}$. Similar to traditional tatonnement, the agent's price at time t is updated based on the difference between the agent's desired reliability, $r_{des,i,t}$, and $r_{act,i,t}$:

$$p_{i,k+1,t} = p_{i,k,t} + \lambda\left(r_{des,i,t} - r_{act,i,t}\right) \tag{10.10}$$

Because a concave utility function is assumed, there is a one-to-one correspondence between $p_{i,k,t}$ and $r_{des,i,t}$, the optimal desired reliability for a given price.

10.3.3 Distributed Decision Making

After receiving the feasibility of its last bid, an agent must make a decision when to place its next bid. Because of the uncertainty in where other agents will bid, an agent cannot only use the most recent prices; there is value in past price information. For example, if distributed agents a and b must choose one time slot $t \in \{1, 2, 3\}$ and select time slots uniformly at random, there is a $1/3$ probability that both will select the same slot and $2/3$ probability that they will not. Assuming that the network can accommodate only one activity at each time step, prices will necessarily increase when nodes schedule themselves together and decrease when they select different slots. This leads to oscillation of prices.

One observation is that there is value in the agent's bidding history. Therefore, agents use the value of past prices to calculate the expected utility of scheduling at a certain time t between lb and ub:

$$\mathrm{E}\left[U_{i,t}\right] = \frac{1}{k} \sum_{m=t-k}^{t-1} U(p_{i,t,m}) \tag{10.11}$$

Agents then use the expected utility to make bidding decisions. To choose the next bidding time, an ϵ-greedy approach works well for deciding whether to choose the time with the maximum expected utility. If agents act greedily and select the time with the highest expected utility, agents risk only bidding during constrained time periods.

Finally, agents must converge to a schedule. To do this, agents send a *decide* bid that signals to the market that they have decided on a time and will not change their time on future bidding rounds. Intuitively, an agent *decides* (i.e. fixes its time bid) when it has reached a suitable confidence level in its expected utility estimates:

$$\sum_t \frac{1}{(k-1)^2} \sum_{j=t-k}^{t} \left(\mathrm{E}\left[U_j\right] - \mathrm{E}\left[U_t\right]\right)^2 < \epsilon_{decide} \tag{10.12}$$

where the left-hand side is a sum over all the variances for each expected utility and the right hand side represents a threshold. Once this threshold is reached, the agent announces to the market that it settled on the time of its maximum expected utility and will not change on future bids.

After an agent fixes its time, it must keep bidding because its reliability may change as other agents converge. Final convergence of the algorithm occurs when

$$\left|r_{des,i} - r_{act,i}\right| < \epsilon_{reliability} \; \forall \; i \in A \tag{10.13}$$

10.3.4 Symmetry Breaking

A final consideration is the need to break symmetry; i.e. identical activities will tend to schedule themselves at the same time, leading to high peak load. One often used method to break symmetry is to introduce randomization [31]. Randomization ensures that loads schedule themselves in a partially random way to make it less likely that they will collide. This algorithm already introduces randomization through its use of ϵ-greedy scheduling. The implementation of the market for reliability algorithm also made use of skipping bidding rounds; this caused the variances in Eq. 10.12 to converge at different rates.

10.4 Description of Algorithm

The complete algorithms are described in Algorithms 10 and 11.

Algorithm 10: Calculation of bidding price and time for each activity i

```
   Input: Activity $A_i = \{lb, ub, P(t), tier\}$
   Output: Schedule = { time , reliability }
1  $\boldsymbol{p_i} \leftarrow \mathbf{0} \; \forall \; lb \leq t \leq ub$
2  $decided \leftarrow false$
3  while $|r_{des,i} - r_{act,i}| < \gamma \cup decided = false$ do
4      if $decided = false$ then
5          $\mathrm{E}[\mathbf{U}] \leftarrow ComputeExpectedUtilities(\boldsymbol{p_i})$
6          $var \leftarrow ComputeVariances(\mathrm{E}[\mathbf{U}])$
7          if $var < \epsilon_{decide}$ then
8              $decided \leftarrow true$
9              $bid.time \leftarrow getMaxExpUtility(\mathrm{E}[\mathbf{U}])$
10             $SendMessage(decided)$
11         end
12         else
13             $bid.time = getRandomizedTime(\mathrm{E}[\mathbf{U}])$
14         end
15     end
16     $bid.price \leftarrow \boldsymbol{p_i}(bid.time)$
17     $SendMessage(bid)$
18     $r_{act,i} \leftarrow ReceiveMessage()$
19     $p_{i,t} \leftarrow UpdatePrice(bid.time, r_{act,i})$
20     $r_{des,i} \leftarrow ComputeOptimalReliability\,(p_{i,t})$
21 end
22 $Schedule.time \leftarrow bid.time$
23 $Schedule.reliability \leftarrow r_{act,i}$
```

10.4.1 Agents Determining Bids

As described in Algorithm 10, agents formulate their bids using the following process. If the agent's desired reliability has not yet converged to the market's allocated reliability (Line 3), there are two possibilities: The agent has fixed its bidding time or the agent has not yet converged to a fixed bid time. If the agent has already converged to a fixed bidding time, it simply receives updates from the central market (Line 15), updates prices based on the received actual reliability (Line 16) and recompute its optimal desired reliability for the next round. If the agent has not yet converged to a time, it must update its expected utilities given the most recent price update (Line 5, where *ComputeExpectedUtilities* implements Eq. 10.11) and determine if the variances have converged (Line 6, where *ComputeVariances* implements Eq. 10.12).

Algorithm 11: Algorithm that allows the central market to determine the feasible reliabilities for all input bids

```
   Input: Set of bids, B, where $b_i$ is as defined in subsection 10.1.2, set of solar
          generation scenarios, S
   Output: Vector of actual reliabilities, $\boldsymbol{r_{act}}$
 1 for $b \in B$ do
 2     $b.dispatched \leftarrow 0$
 3 end
 4 for $k < TotalIterations$ do
 5     for $s \in S$ do
 6         $\boldsymbol{power_{cur}} \leftarrow s$
 7         for $b \in \boldsymbol{B}$ do
 8             $b.marked \leftarrow false$
 9         end
10         for $b \in \boldsymbol{B}$ do
11             $b_i \leftarrow SelectUnmarkedActivity(\boldsymbol{B.price})$
12             if $\boldsymbol{power_{cur}} - b.P(t) > 0 \; \forall \; t \in b.P(t)$ then
13                 $b.dispatched \leftarrow b.dispatched + 1$
14                 for $t \in b.P(t)$ do
15                     $power_{cur}(t) \leftarrow power_{cur}(t) - b.P(t)$
16                 end
17             end
18             $b.marked \leftarrow true$
19         end
20     end
21 end
22 for $b \in \boldsymbol{B}$ do
23     $r_{act,i} \leftarrow b.dispatched/(TotalIterations * |\boldsymbol{S}|)$
24 end
25 return $\boldsymbol{r_{act}}$
```

If the variance have converged, then the agent will finalize its bid time (Line 9). If they have not yet converged, the algorithm will pick a randomized bid time (Line 12). Note that if all prices are initialized to the same variable, then all variances will immediately converge. A simple method to avoid this is to wait until the variances go from increasing to decreasing before running Line 6.

10.4.2 Market Determining Feasibility

The purpose of Algorithm 11 is to determine the actual reliability of each bid b by running enough simulations of all generation scenarios. For each generation scenario, it does this by selecting activities based on their bidding price (Line 9 which selects activities based on Eq. 10.8). A selected activity is then attempted; if there is enough power remaining in the current generation scenario for all times when the activity would like power, then it is dispatched and the number of times an activity is dispatched is increased by one (Line 11). In either case, the activity is marked (Line 14) to ensure it cannot again be picked during the current simulation. From Line 10, that there must be enough power to serve a bid during its entire span ($\forall t \in b.P(t)$), which means that dispatching a fraction of a bid is not allowed. The total number of times an activity is dispatched is then divided by the total number of simulations run, to give the final frequency for all bids, $\boldsymbol{r_{act}}$.

10.5 Proof of Convergence

In the following section a proof of convergence is presented, where convergence is defined by Eq. 10.13. In normal tatonnement, convergence can be proved through the use of a potential function argument or through relating tatonnement to dual decomposition where the prices serve as the Lagrange multiplier [92]. However, because there is not a price per good but a set of prices for one commodity, reliability, the proofs are not obvious to translate to this case.

First a lemma is proved related to all activities fixing their time bids.

Lemma 1. While deciding on a scheduling time, each agent's vector of expected utility converges to a vector of constant values, $\boldsymbol{E}\left[U_i\right] \rightarrow \boldsymbol{c}$.

Proof. This is proved by showing that the prices at each time step form a recurrent Markov chain with a limiting distribution and therefore have a well-defined expectation.

For an agent i at a time step t with bidding price $p_{i,t}$, let $\boldsymbol{p_{i,t}}$ represent the vector of all possible prices. This vector is finite because we can apply an arbitrarily fine discretization to the bidding prices (or round to the closest cent) and that the prices are bounded above and below because of the utility (at some point, if prices rise to the extent that agents receive negative utility, then the agent will not increase its bid further). Whether agent i transitions from one price $p_{i,t,k}$ to another $p_{i,t,k+1}$ is dictated by which other activities bid at time step t and the prices of the other bids. Therefore, a second vector is created $\boldsymbol{p'_{t,k}}$ that contains agent i's prices and every other possible combination of other agents prices for all $a \in A$ at round k. Similar arguments for $\boldsymbol{p_{t,k}}$ imply that $\boldsymbol{p'_{t,k}}$ is finite.

Let $\boldsymbol{p'_{t,k}}$ be the states of a Markov chain. For each individual state, $p'_{t,k}$, that is composed of the prices of all agents who may bid at time t, the transition from one state to the next is well-defined and depends only on which activities bid at time t at round k. Therefore, we define the transition probabilities for the Markov chain as the probability that a certain subset of activities $a \in A$ bids.

A Markov chain has a stationary distribution π if all states are positive recurrent, where positive recurrence means that the expected time between being in state s and returning to that state is less than infinity. The Markov chain described above is shown to be positive recurrent. To simplify the argument, it is assumed that all prices $p_{t,i,k}$ are initialized to the price that agent i would receive if only agent i bid at time step t. Let B be an ordering of bids that causes $p_{i,t,k}$ to take the value $p_{i,t,k+n}$. To return to $p_{i,t,k}$, it is only necessary that the reverse of B occur, $\overline{B}$. This is possible because for auction k, all activities have the option to bid. Because $p_{i,t,k+n}$ is arbitrary, this is true for any price. Therefore, the bidding prices form a positive recurrent Markov chain with stationary distribution π_t. The constant c_t is simply the expected value of π_t: $c_t = \frac{1}{N}\sum_n \pi_{t,n}$. □

This completes the proof that each agent's expected utility vector converges. This allows each agent to decide on a scheduling time. Next, it is shown that once all agents have decided, all agents' desired reliabilities converges to their actual reliabilities. To simplify the proof and avoid the issue of how to tune λ in Eq. 10.10, we prove convergence for the simple bidding strategy where the magnitude of all price updates equals some small constant ϵ. It is also assumed that if $|r_{des,i} - r_{act,i}| < \delta$ for some small constant δ, then agent i does not update its price at bidding round k.

First, a number of lemmas are stated.

Lemma 2. For all activities, $\frac{dr^\star_{des}}{dp} < 0$ where $r^\star_{des}$ is the optimal desired reliability given a price p.

This follows directly from the fact convex and monotonically increasing utility functions are used.

Lemma 3. If a_i is of a lower reliability tier then a_j, then $\frac{dr^\star_i}{dp_i} \leq \frac{dr^\star_j}{dp_j}$

Qualitatively, Lemma 3 means that a higher tiered activity will bid more for greater reliability. Mathematically, this follows from the different utility curves; in the simple case of quadratic utilities, $U(r,p) = ar^2 + br + c - pr$, the optimal reliability given a price occurs at $\frac{b-p}{2a}$. Utilities of a higher tier have a larger constant b, implying they will bid more for reliability.

Lemma 4. If all agents change their bid by no more than a small constant ϵ, then there also exists a small constant γ such that their actual reliabilities, $r_{act,i}$, change by no more than γ.

This is shown for the case of two activities, a_1 and a_2. At round k, a_1 is selected over a_2 with probability $\frac{p_1}{p_1+p_2}$. If a_1 increases its price by no more than $|\epsilon|$, a_1 is now selected with probability $\frac{p_1+\epsilon}{p_1+p_2+\epsilon}$. Note that the variables ϵ can be made arbitrarily small so that a_1's overall reliability does not increase by more than γ.

Lemma 5. For a bidding strategy where all price increases /decreases equal a suitably small constant $|\epsilon|$, after a large enough number of rounds K, $|r_{des} - r_{act}| < \delta \; \forall \; i \; \in A$ for some small constant δ and the algorithm converges.

Proof. Assume that all prices are initialized to 0, so that all initial desired reliabilities equal 1. Because all prices are initially equal, all initial actual reliabilities, $r_{act,0}$ are also equal. Now, let each agent increase prices by ϵ so that after k rounds $p_{i,k} = \sum_k \epsilon$. Prices will continue to increase until the agents in the lowest tier L reaches a point when $|r_{des,k} - r_{act,0}| < \delta$. This happens only for the lowest tier by Lemma 3. At this point, agents in L will stop increasing their price. Because of the differences in bid prices, agents in L will receive lower r_{act} on the next round and agents not in L will receive higher r_{act}. However, by 4, no change in r_{act} will be more than γ and therefore $r_{des} > r_{act}$. If L's r_{act} decreases enough, agents in L will continue to increase prices. This process will repeat itself for all tiers with lower tiers not increasing their prices on rounds when $|r_{des,k} - r_{act,k}| < \delta$. Consequently, for this simple bidding strategy, there are the following properties of r_{des} and r_{act}:

- r_{des} are monotonically decreasing as price either increases by ϵ or does not change if $|r_{des,k} - r_{act,0}| < \delta$
- r_{act} either monotonically decreases (for lower tiers) or monotonically increases (for higher tiers) and is bounded by 0 and 1.
- Some price must change if there exists an i for which $|r_{des,i,k} - r_{act,i,0}| > \delta$, so the algorithm will not complete with any $|r_{des,i,k} - r_{act,i,0}| > \delta$.

These three facts imply that after a sufficient number of rounds, we will have $|r_{des,k} - r_{act,0}| < \delta$ and the algorithm converges. □

The bidding strategy of incrementally increasing prices by an infinitesimal amount is overly conservative. In experiments, λ values between 0.1 and 0.5 are used and there were not issues with convergence.

10.6 Test Case for Market for Reliability Algorithm

A test case was performed to illustrate the algorithm. The test case consisted of twelve activities of two types: lighting and cell charging (typical activities for Indian villages). Each type of activity had the following time constraints and power characteristics:

1. **Charging** = $\{lb = 0,\ ub = 12,\ duration = 3,\ power = 5\ \}$
2. **Lighting** = $\{lb = 6,\ ub = 12,\ duration = 2,\ power = 8\ \}$

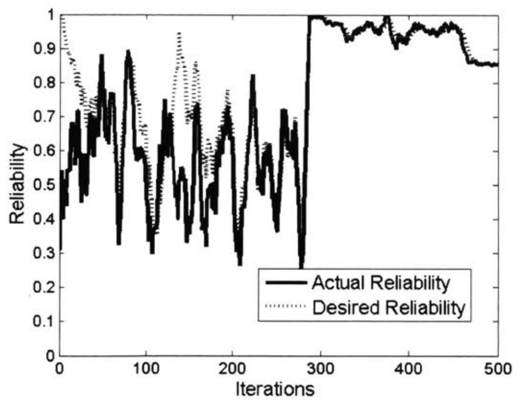


Figure 10-3: Desired reliability and actual reliability for one activity converging.

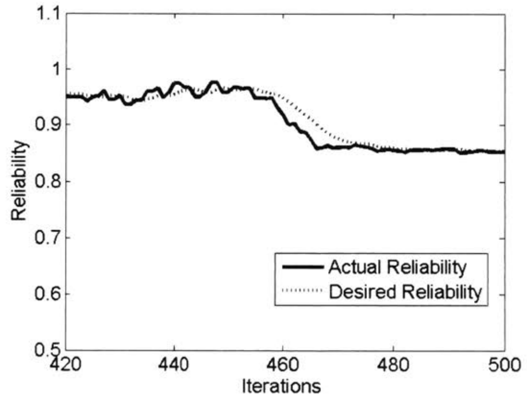


Figure 10-4: Convergence of one activity once all activities have decided on a time.

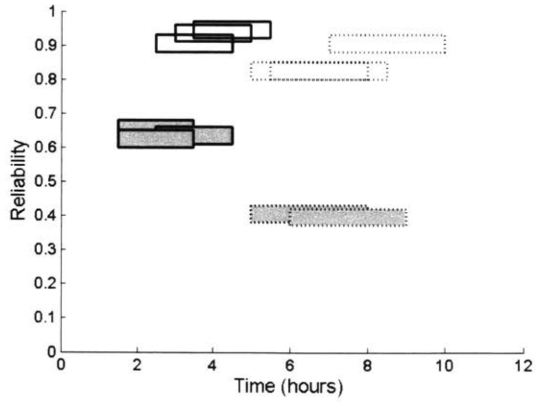


Figure 10-5: Results for a 12-activity test case. Tier 1 activities have no fill and Tier 2 activities are filled in gray. Cell charging has solid outlines and lighting activities are dashed outlines.

Activity	1	2	3	4	5	6	7	8	9	10	11	12
Type	C	C	C	L	L	L	C	C	C	L	L	L
Tier	1	1	1	1	1	1	2	2	2	2	2	2
Frequency (dMILP)	0.90	0.79	0.58	0.94	0.58	0.87	0.67	0.55	0.73	0.48	0.33	0.44
Frequency (M.f.R.)	0.96	0.93	0.97	0.85	0.85	0.93	0.68	0.66	0.65	0.43	0.43	0.42

Table 10.1: Results of scheduling a 12 activity, 2-tier network for the deterministic Mixed Linear Integer Program (dMILP) and the Market for Reliability (M.f.R.) algorithm. C represents a charging activity and L represents a lighting activity. Frequency represents the probability an activity was served.

The scheduling horizon was a twelve-hour day. Cell charging could take place over the entire day while lighting could only take place at night. Activities were divided into two tiers, where Tier 1's activities were served with the greatest probability. The utility functions were quadratics, similar to those in Figure 10-2.

Selected results are shown above. Figure 10-3 shows the actual and desired reliabilities plotted over one run of the algorithm. During initial bidding rounds, there is a large amount of fluctuation as the agent bids during various time steps. Once other agents begin to converge, the uncertainty reduces until finally, when all agents have decided on a time, the desired reliability and actual reliability converge. The final convergence is shown in 10-4. Figure 10-5 shows the output schedule. Tier 1 activities receive higher reliability than Tier 2 activities. Although they are unconstrained in time, agents uniformly decided to schedule cell charging activities during the day because lighting activities are constrained to only happen in the evening and must compete for energy.

10.7 Benchmarking Against a Deterministic Approach

The Market for Reliability algorithm is benchmarked against a deterministic approach where the scheduling and dispatching are done on a deterministic basis. The problem is reformulated as a deterministic MILP, comparable to the approach taken in [117]. A deterministic MILP is selected rather than a stochastic program because deterministic MILPs are most often used in industry for power scheduling [72] and a stochastic program would be unreasonable given the strict memory requirements for low cost controllers. For a deterministic approach, instead of a distribution over the random variables, only one generation scenario is considered, taken as the expected value of the generation:

$$g(t) = \mathbf{E}[\mathbf{S}] = \sum_{s} p(s)\, g_s(t) \tag{10.14}$$

Scheduling is over a discretized time horizon $\mathbf{T} = [t_1, ..., t_N]$ and uses binary decision variables $x_{i,t}$ which are 1 if Activity i receives is scheduled for time period t. Binary variables are also used for the start times of activities, $y_{i,t}$.
The objective is a linear combination of the y decision variables, indicating whether an activity was served or not:

$$\max \sum_{i,t} c_i y_{i,t} \tag{10.15}$$

where c is a constant that relates to the benefit of serving a certain activity (i.e. this could relate to tiers of service).

The following constraints ensure that an activity persists for a specified duration after its start time:

$$y_{i,t} \implies \bigwedge_{k=t}^{t+Z_i} x_{i,t+k} \tag{10.16}$$

where Z_i is the length of activity i. To ensure that each activity is only scheduled once we have:

$$\sum_{t}^{N-Z_i} y_{i,t} = 1 \tag{10.17}$$

There are the following energy balance constraints:

$$\sum_{j \in G} g_{j,t} - \sum_{i \in L} c_{i,t} x_{i,t} + W_t = 0, \forall t \in N \tag{10.18}$$

where $g_{j,t}$ is the energy generated by the jth generator and is a deterministic

Method	Utility During Execution
dMILP	56.2
M.f.R.	64.7

Table 10.2: Comparison of utilities between the two approaches

constant. The constant $c_{i,t}$ represents the amount of power required by activity i and W_t is the amount of energy that is neither used for an activity and is effectively wasted.

10.7.1 Comparison of Results

The MILP was solved using Gurobi on a PC with 8 GB of RAM and an Intel i7 Processor at 3.4 GHz [41]. The scheduler was run 5000 times and the overall frequency with which each activity was dispatched recorded. Each algorithm was compared using the same stochastic uncertainty in solar generation. For the deterministic MILP, this meant generating a schedule using the generation's expected value (Eq. 10.14) and then dispatching activities only if there was enough power to serve them in a stochastic simulator. Similar to the Market for Reliability algorithm, only completed activities were recorded as a success. The results of the comparison are shown in Tables 10.1 and 10.2. Table 10.1 shows that both algorithms dispatched the higher tier activities more often, with the M.f.R. algorithm dispatching Tier 1 activities with a higher frequency. Table 10.2 shows the difference in utility between the two approaches. A Tier 1 activity was given 10 units of utility if dispatched and Tier 2 utility only 3 (i.e. these were the utility values input into the MILP's utility function, Eq. 10.15). Table 10.2 shows that the M.f.R. algorithm reaches a better solution than the corresponding MILP formulation.

Chapter 11

Centralized Resource Allocation

While the approach in the previous chapter was able to solve the resource allocation problem in a distributed manner, it had high communication complexity. Messages were needed to relay bidding information between the agents and central market. This could be challenging to implement on low cost hardware and cause the algorithm to take a long time to converge.

This chapter focuses on a centralized planner. Rather than using a Monte Carlo simulation as in the previous chapter, this chapter models the state distribution using a discretization. The resource allocation problem is posed as a Markov Decision Process and a stochastic local search used to compute good policies.

11.1 Innovations

Work in this chapter sits between techniques in stochastic programming and goal-directed, chance-constrained planning. A stochastic resource allocation algorithm is developed that receives statistical models of resource availability and demand and outputs a user-specific control policy. Specifically, we offer the following contributions:

1. The ability to reason directly over distributions that describe user goals allows direct integration with end users and the ability to directly shape user behavior, unlike most approaches in literature.

2. Modeling the stochastic evolution of the resource as a discrete time and state Markov chain allows a memory-efficient representation of state stochasticity.

3. A conflict-based stochastic local search method designed for resource allocation efficiently explores the search space and obtains solutions that are close to optimal.

11.2 Stochastic Resource Allocation Problem

The problem is formalized as a Chance-Constrained Stochastic Resource Allocation Problem (*ccSRAP*). A *ccSRAP* is an 8-tuple $P = \langle T, G, \mathcal{U}, \mathcal{A}, \mathcal{C}, SC, \Pi, f_u \rangle$ where:

- T is the planning horizon, with time discretized.
- G is a distribution describing resource availability or generation, defined over T: $supp\,(G) = \{t | 0 \leq t \leq T\}$.
- $\mathcal{U}$ is a set of users.
- $\mathcal{A}$ is a set of resource-consuming activities. Each activity $a \in \mathcal{A}$ is associated with a user. Corresponding to each activity is a probability distribution $f\,(a)$ that describes the user's resource consuming preferences such as time-of-use or the amount of resource required.
- $\mathcal{C}$ is a set of chance constraints that enforce the probability of serving certain critical activities. Each chance constraint c_i is associated with one activity distribution $f(a)$. The set of activities associated with chance constraints, $\mathcal{A}_c$, is a subset of all activities $\mathcal{A}_c \subseteq \mathcal{A}$, i.e. not all activities are associated with chance constraints.
- SC is a set of stochastic constraints that direct the evolution of the system state s, i.e. the availability of a resource over time.
- Π is a set of control policy functions, π_a, where each policy π_a is associated with an activity distribution $f(a) \in \mathcal{A}$. Each function π_a maps the system state s

to a set of time bounds M_π that describe a's allowed start time and duration: $M_{\pi,a} = \pi_a(s)$.

- f_u is an objective function that maps the set of policies to positive utility $U \in \mathbb{R}^+$, $U = f_u(\Pi)$.

Each component is described more fully in the following sections.

11.2.1 Model of Time

The algorithm reasons over a specified time horizon T where time is discretized into Δt. A natural time horizon to use for electricity scheduling is 24 hours because of the repetitive nature of generation and demand.

11.2.2 Input Model of Demand and Generation

The input is the joint distribution of resource demand D and resource availability or generation G:

$$f(G, D) \tag{11.1}$$

To simplify modeling of (11.1), independence is assumed between demand and generation. Further, the naive assumption is made of independence between individual users and their demanded activities:

$$f(G, D) = f(G)\, f(D) = f(G) \prod_a f(a) \tag{11.2}$$

To model $f(G)$, a sample average approximation (SAA) is used. Sample average approximation is commonly used in stochastic optimization problems and requires sampling resource scenarios from their distribution [62] where a single scenario i is a vector: $G_i = [g_{i,0}, g_{i,1} \ldots, g_{i,T}]$. The distribution can be generated through historical or sensor data, such as observing the solar generation available on a microgrid.

11.2.3 Activity Model of Demand

Resource demand is modeled in terms of activities that users would like to accomplish. Uncertainty is modeled through continuous distributions, which are parameterized by statistics relating to the user's desired start time, duration, and resource demand for the activity:

$$f(a) = f(start\_time, duration, resource) \tag{11.3}$$

For the microgird application, constant power direct current (DC) activities are assumed such as cell charging and lighting. Constant power is reasonable in the developing world context [49]. In the case where users' preferences are uniformly distributed, (11.3) can be rewritten as a lower bound and upper bound on each of the uncertain quantities:

$$f\left(st, dur, r\right) \sim \mathcal{U}\left(t_{st,des}, t_{dur,des}, r\right) \tag{11.4}$$

$$\text{where } \; t_{st,des} \in \left[t_{st,lb,des}, t_{st,ub,des}\right] \tag{11.5}$$

$$t_{dur,des} \in \left[t_{dur,lb,des}, t_{dur,ub,des}\right] \tag{11.6}$$

$$r = r_{power} \text{ (a constant)} \tag{11.7}$$

A distribution of the form (11.4) is input to the algorithm as a model of the users' preferences. Because it represents the user's desires with regards to the start times and duration of the activity, it is referred to as the *preference distribution.* While this work assumes distributions with finite support (such as the uniform), distributions with infinite support can be truncated to match this model.

11.2.4 Output Control Policy

The algorithm outputs a set of control policies Π, which is a relationship between a state-action pair defined over the time horizon T. The overall policy Π is a set of independent policies for each individual activity a:

$$u_a\left(s\right) = \pi_a\left(s(t)\right) \; \forall \; \pi_a \in \Pi \tag{11.8}$$

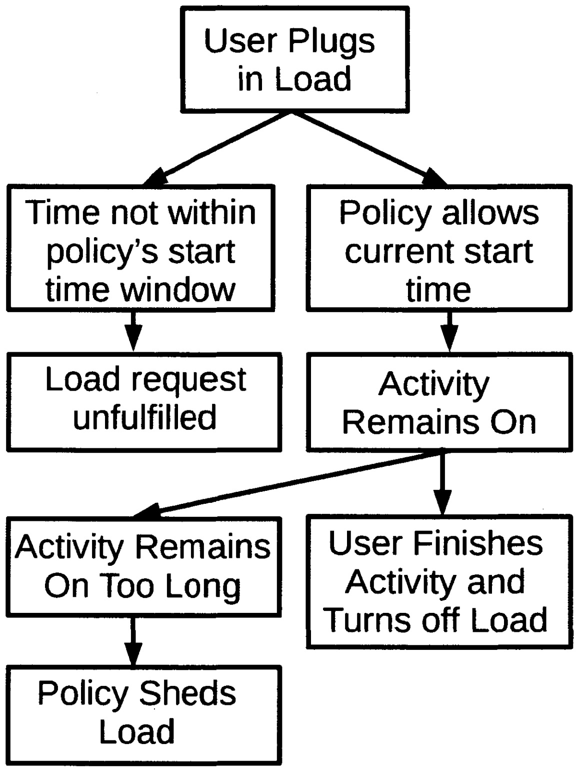

User Plugs in Load
Time not within policy's start time window
Policy allows current start time
Load request unfulfilled
Activity Remains On
Activity Remains On Too Long
User Finishes Activity and Turns off Load
Policy Sheds Load

Figure 11-1: Depiction of the controller states in serving electrical activities for the microgrid use case

In the microgrid example, the state s is a function of the battery's state of charge $b(t)$ and the estimated amount of solar energy remaining, $\mathbb{E}[G(t)]$. Each control policy is unique to an activity, while the state s is network-wide. The action $u_a(s)$ is whether to supply an activity with the requested resource C_{req}, depending on whether the resource is requested within precomputed time windows M_π:

$$u_a(s) = \begin{cases} C_{req} \text{ for } t \in M_{\pi,a} = [t_{lb,\pi}(s), t_{ub,\pi}(s)] \\ 0 \quad \text{ for } t \notin M_{\pi,a} = [t_{lb,\pi}(s), t_{ub,\pi}(s)] \end{cases} \tag{11.9}$$

Eq. 11.9 is further divided into two separate policies $u_{start,a}$ and $u_{dur,a}$, controlling an activity's allowed start times and durations respectively:

$$u_{start,a} = \pi_{start,a}(b(t), \mathbb{E}[G(t)]) \mid a \text{ not started} \tag{11.10}$$

$$u_{dur,a} = \pi_{dur,a}(b(t), \mathbb{E}[G(t)]) \mid a \text{ started} \tag{11.11}$$

Eq. 11.10 represents the policy given that the activity has not yet started, i.e. it governs the activity's allowed start times conditioned on the network state. Eq. 11.11 governs the activity's duration. Creating policies for an activity's start time and duration provides more control authority. There may be cases when it is advantageous to shift an activity (i.e. change its start time) or other occasions when it is necessary to curtail its resource usage (i.e. change its duration). The control strategy is depicted in Fig. 11-1.

11.2.5 Objective Function

While many electricity resource allocation problems are formulated using objectives that focus on minimizing grid cost, this work proposes a user-centric objective. The objective seeks to maximize the probability that the output policy set Π allows users to receive a resource when and in the quantity that they want:

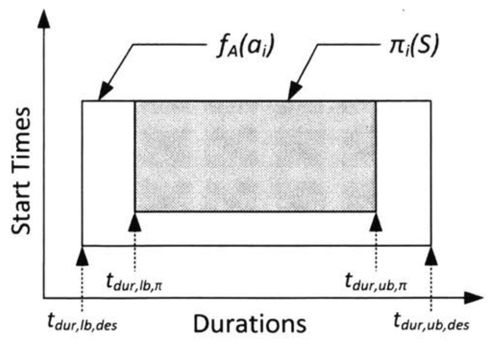


Figure 11-2: The objective seeks to minimize the difference between the policy π_a's output time bounds and the user's preference distribution $f(a)$, shown here for a uniform distribution over start times and durations.

$$\max_{\Pi} \sum_{\mathcal{A}} \sum_{supp(f(a))} P\left(a \in M_\pi | a, \pi_a\right) P\left(a\right) \tag{11.12}$$

$$- c_1 \sum_{crit.} \mathbb{1}_{U_a \leq U_{a,min}} \tag{11.13}$$

where $P\left(a\right)$ represents the probability of a specific realization of a.

Maximizing the Likelihood of Achieving User Goals

The first term, (11.12), seeks to determine the policy that maximizes the likelihood of achieving user goals, conditioned on users' preferences and the control policy, π_i. For a grounded example, assuming a user wants 5W of power starting between 1 PM and 3 PM with a duration between 2 to 3 hours, the best policy is one that allows all start times and durations over the distribution's support with probability 1 (i.e. the user may start whenever they want and consume their desired amount of electricity almost surely). The approach is illustrated in Fig. 11-2.

The second term, $P\left(A = a\right)$, indicates that utility is weighted by the preference distribution $f(a)$, i.e. the probability the user choses a specific realization of a. The inner summation is over the support of the user's preference distribution (no utility is gained by providing a resource when a user does not want it). The first term,

$P\left(a \in M_\pi | A = a, \pi_a\right)$, is the probability a specific activity realization is successful conditioned on the policy. It is more difficult to compute. First, the term is conditioned on the resource's distribution, $P(s)$:

$$\begin{aligned} P\left(a \in M_\pi | A = a, \pi_a\right) = & \\ & \int_s P\left(a \in M_\pi | A = a, \pi_a, s\right) P(s) ds \approx \\ & \qquad \sum_S P\left(a \in M_\pi | A = a, \pi_a, S\right) P(S) \quad (11.14) \end{aligned}$$

Where the approximation to the integral is made possible by a discretization of the continuous state s. To compute Eq. 11.14, it is necessary to determine $P\left(S\right)$, which is covered in the following section.

Chance Constraints on Activity Reliability

The second term in the utility, line 11.13, is a penalty term for not serving a minimum utility to activities contained within $\mathcal{C}$. $\mathcal{C}$ is a set of pre-specified critical activities that should be served with a lower bound on reliability. This is useful to provide quality of service guarantees in the face of stochastic resource availability. A chance constraint on a critical activity takes the form:

$$P\left(U_{a,crit,min}\right) \geq 1 - \epsilon \;\; \forall \;\; a \in \mathcal{C} \qquad (11.15)$$

where $U_{a,crit,min}$ is a minimum utility that activity a is guaranteed with probability $1 - \epsilon$. Because probability mass serves as a proxy for utility, Eq. 11.15 is a guarantee on the likelihood a critical activity is served. It is written generally because there are many possible forms the minimum utility, $U_{a,crit,min}$, could take. For example, reliability guarantees may be written with respect to activity start times, durations, or both. For this work, it is assumed reliability constraints are specified with regards to an activity's duration (such as guaranteeing two hours of lighting with 90% reliability over all desired start times). This specification can be translated into the minimum

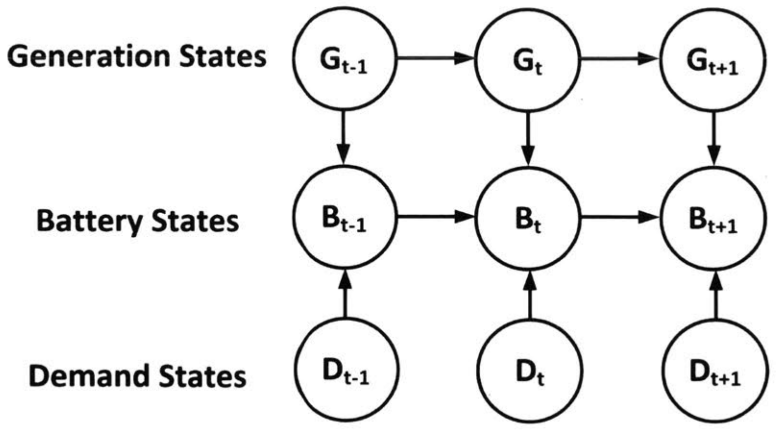


Figure 11-3: Modeling uncertainty in the resource state as a Markov Decision Process.

utility:

$$U_{a,crit,min} = P\left(a_{crit,min}\right) = \sum_{S} P\left(dur \geq dur_{min} | S, \pi\right) P\left(S | \pi\right) \geq 1 - \epsilon \quad (11.16)$$

The minimum utility requirement, Eq. 11.16, is included in the objective function as a barrier term and ensures the solution incurs a penalty for each violated chance constraint. The reason that constraint violation is included as a barrier term and not a hard constraint is that the algorithm must always return a solution. It may be impossible to satisfy Eq. 11.15 (for example, if solar generation is constantly poor during a rainy season). In this case, the algorithm will maintain the reliability guarantees on a best effort basis.

11.3 Approximating Complex Scenarios as an MDP

To compute the utility of a policy, it is necessary to compute the distribution of the system state as it evolves in time. An efficient method to model stochastic state evolution is a discrete time and state Markov Decision Process (MDP). The MDP formalism consists of the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$: states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities $\mathcal{T}$, and a reward function $\mathcal{R}$. The MDP formalism is used to model the $ccSRAP$;

the components are described in the following subsections.

States $\mathcal{S}$: Resource Availability

Because of the vast number of generation and demand scenarios, modeling and reasoning about the system state is challenging. The Markov assumption combined with a state discretization makes this reasoning task much more tractable. Instead of reasoning about a large number of resource scenarios, it is necessary to consider the resource state at time $t-1$:

$$P\left(S_{t+1}|S_t, S_{t-1}, \ldots, S_{t=0}\right) = P\left(S_{t+1}|S_t\right) \tag{11.17}$$

To map from scenarios to discrete states, a discretization is applied. Each continuous resource state falling within a predetermined bound, $\{s \in \mathbb{R}^n | s_{lb} \leq s(t) < s\}$, is mapped to a discrete state $S_{t,i}$. As mentioned above, in the microgrid case, the resource state is the amount of battery energy and expected solar generation available at time step t. Each discrete state S represents a set of continuous battery energies and solar powers, $S_{t,i} = \{b_t, \mathbb{E}\left[G_t\right] \mid b_{lb,i} \leq b\left(t\right) < b_{ub,i}, G_{lb,i} \leq \mathbb{E}\left[G\left(t\right)\right] < G_{ub,i}\}$.

Transitions Probabilities $\mathcal{T}$: Changing Resource States

To compute $P\left(S_{t+1,j}|S_{t,i}\right)$, the probability of moving from one discretized resource state to another, it is necessary to consider all combinations of activities and generation scenarios that would cause a transition from $S_{t,i}$ to $S_{t+1,j}$. For the microgrid case, the transition probability is computed as

$$P\left(S_{t+1,j}|S_{t,i}\right) = P\left(G_{i,j}\right) P\left(b_{i,j}|\mathcal{A}, G_j\right) \tag{11.18}$$

In Eq. 11.18, $P\left(G_{i,j}\right)$ is the probability of transitioning from one solar generation scenario to another, e.g. moving from a sunny day to a cloudy day. The second term, $P\left(b_{i,j}|\mathcal{A}, G_j\right)$, is the probability of the battery state b changing due to resource generation and consumption. It is computed as:

$$P\left(b_{i,j}|\mathcal{A}, G_j\right) = \sum_{\mathcal{E}\in C_{i,j}} P\left(\mathcal{E}\right) \tag{11.19}$$

where $\mathcal{E}$ represents a single scenario of demand and generation that causes a transition from b_i to b_j and $C_{i,j}$ is the set of *all* such scenarios (there are potentially many such scenarios since the state is discretized). The scenario $\mathcal{E}$ is described by the following constraints:

$$b_{t+1,j} = b_{t,i} + \sum_{\mathcal{A}\in\mathcal{E}_i} r_t\left(a\right) + g_{i,t} \tag{11.20}$$

$$\text{s.t. } b_{t,i,lb} \leq b_{t,i} < b_{t,i,ub} \tag{11.21}$$

$$b_{t+1,j,lb} \leq b_{t+1,j} < b_{t+1,j,ub} \tag{11.22}$$

where (11.20) is the energy balance, $r_t\left(a\right)$ the power consumed by activity a at time t, and $g_{t,i}$ the energy generation.

Actions $\mathcal{A}$: Altering Activity Time Windows

Unlike traditional MDPs, $P\left(S_{t+1,j}|S_{t,i}\right)$ cannot be directly altered because there is not direct control over resource generation or consumption. As indicated by Fig. 11-1, users are free to demand resources when they choose, but the policy enforces time windows that prevent consumption outside of those windows. The controller's actions affect the probability that a user is consuming power at time t, i.e. the policy affects the probability of scenario $\mathcal{E}$. This distribution is the probability of the generation scenario g_i multiplied by the probabilities of activities $a \in \mathcal{E}$ consuming the resource at time t:

$$P\left(\mathcal{E}\right) = P\left(g_i\right) \prod_{a\in\mathcal{E}_i} P\left(a|\pi_a\right) \tag{11.23}$$

The probability of a consuming resource at time t can be written in terms of the sums:

$$P\left(a|\pi_a\right) = \sum_{t_0,st}^{t_f,st} \sum_{t_0,dur}^{t_f,dur} P\left(t_{dur,des}|t_{st,des},\pi_a\right) P\left(t_{st,des}|\pi_a\right) \tag{11.24}$$

where the outer sum is over all possible policy-allowed start times that would result in the activity being on at time t and the inner sum is over all possible policy-allowed durations that would result in the activity consuming the resource at time t. Clearly, a policy that allows more start times and durations cannot decrease $P\left(a|\pi_a\right)$, a fact which is used later in designing the stochastic local search algorithm.

Rewards $\mathcal{R}$: States with Positive Resource

For resource allocation problems, demand can be served when resources are available. For electricity scheduling, this means that transitioning to a state with positive battery energy receives a reward. The expected reward over all states is written in terms of this:

$$\mathbb{E}\left(\mathcal{R}\right) = \sum_{t} \sum_{S_t} \sum_{\mathcal{A}} U\left(a,\pi\right) P\left(S_t\right) \mathbb{1}_{b_t>0} \tag{11.25}$$

where $\mathbb{1}_{b_t>0}$ is an indicator function which is 1 if the state has positive battery energy and 0 otherwise. $U\left(a,\pi\right)$ is the utility gained by users being able to receive resource in state S_t. From Eqns. (11.12, 11.14), it is equal $P\left(a \in M_\pi|A=a,\pi_a,S\right) P\left(A=a\right)$, i.e. users only gain utility when the policy allows it and more utility is gained when the user is more likely to desire the resource.

11.3.1 Recurrent Planning Horizon

To fully specify the MDP, a planning horizon T must be specified. This chapter's approach is to impose an initial distribution on states and determine the optimal policy for all states less than T time steps into the future. It is observed that in certain domains, such as electricity scheduling, demand is highly cyclical [68]. Most activities repeat on a daily or weekly frequency. This is used to compute the long term probability that the system is in state S_i over a repeating time horizon T_r. The state distribution at t_0 is the same as t_T, $S_{0,i} = S_{0,T_r}$. This creates a recurrent Markov chain with the transition matrix defined by $\mathcal{T}$. To obtain the long term state probability, the Markov chain's stationary distribution is computed. This is done using (with a slight abuse of notation for π, in this case the stationary distribution π^s):

$$\pi^s = \pi^s \mathcal{T} \tag{11.26}$$

subject to:

$$\sum_{S_t} \pi_t^s = 1 \ \forall \ t \in T \tag{11.27}$$

This system of linear equations can be solved efficiently using the conjugate gradient method or LU decomposition [65]. Solving π_t^s for all time steps results in the long term probability that the system is in state $S(b, \mathbb{E}[G], t)$.

11.4 Conflict-Based Stochastic Local Search

A policy must be output that maximizes terms 11.12 and 11.13. Instead of using a computationally-intensive solver, a conflict-based stochastic local search strategy is used. The approach generates a policy keeping track only of the transition matrix, current policy, an incumbent best policy, and a set of conflicts. This allows generation of a good solution in a very memory-limited environment.

The insight into the search problem is that poor policies are easy to generate and can serve as bases for better policies. A further insight is that poor policies can be improved simply by increasing the policy's time bounds. This suggests the following simple strategy for generating a good policy: first, start with a set of policy bases Π_b, one for each activity. Second, incrementally improve the bounds of each policy until it is no longer possible to do so. The following theorem, stated without proof, guides this strategy:

Theorem 1. Given a set of activities $a \in \mathcal{A}$ and a set of base policies Π_b, increasing the time bounds of any individual activity monotonically decreases the distance

Algorithm 12: Conflict-Based Stochastic Local Search.

```
   Input: Model of generation, $G$
   Input: Set of distributions describing activities, $D$
   Output: Policy $\Pi$ for all activities
1  for max number of iterations do
2      $pol\_proposed \leftarrow InitializeBasePolicy(D)$
3      $cur\_utility \leftarrow CalcUtility(pol\_proposed)$
4      while $pol\_proposed.IsImprovable()$ do
5          foreach $S_{t,i} \in \mathbf{S}$ do
6              foreach $\pi_a(S)$ in $\Pi$ do
7                  $pol\_proposed.Expand()$
                    $new\_utility \leftarrow CalcUtility(pol\_proposed)$
8                  if $new\_utility < cur\_utility$ then
9                      $pol\_proposed.record\_conflict()$
10                     $pol\_proposed.rollback()$
11                 end
12                 else
13                     $cur\_utility \leftarrow new\_utility$
14                 end
15             end
16         end
17     end
18     if $CalcUtility(pol\_proposed) \geq CalcUtility(incumbent\_policy)$ then
19         $incumbent\_policy \leftarrow pol\_proposed$
20     end
21 end
```

between the utility of Π_b and the Pareto optimal frontier, with Pareto optimality defined in terms of the utilities of each a.

The theorem is guided by the fact that improving the time bounds of one activity's policy will never decrease the utility obtained by that activity. It may decrease the utility of other activities by reducing resource availability. Because of this, after each incremental improvement, the policy's global utility is calculated. If the utility has decreased, a conflict is extracted, which prevents further exploration of this region of the state space. To explore the state space, the search is made stochastic by randomizing which time bounds are expanded at each iteration. Finally, because the local search can become stuck in local optima, the inner search is embedded in an outer loop which maintains an incumbent best policy. The complete algorithm is provided in Alg. 12.

The inner loop first initializes base policies, which take the form: $\pi^0_{st,a} = t$, s.t. $s_{lb} \leq t \leq s_{ub}$, $\pi^0_{dur,a} = \delta t$. The initial start time policy is one start time randomly drawn from the preference distribution. The initial duration is the minimum possible duration, i.e. one discrete time step. Given initial policies $\pi^0_{st,a}$ and $\pi^0_{dur,a}$, local improvements are performed that either add or subtract a time step to the proposed policy's allowed start times or duration:

$$t_{st,lb,k+1} = t_{st,lb,k} - \Delta t \tag{11.28}$$

$$t_{st,ub,k+1} = t_{st,ub,k} + \Delta t \tag{11.29}$$

$$t_{dur,ub,k+1} = t_{dur,ub,k} + \Delta t \tag{11.30}$$

The goal is to improve each policy π_a such that the global utility of the policy Π increases at every iteration. If the global utility does not increase, a conflict is extracted. A conflict is a constraint that prevents exploring the search region that caused a decrease in utility. Conflicts guide the proposed policy until they prevent any further expansions or the proposed policy reaches the preference distribution's boundary. When the proposed policy can no longer be improved, its utility is compared against the incumbent and a new iteration begins.

During implementation, further search optimizations are used such as exploring the lowest energy states first and only updating the region of the transition matrix affected by an expansion.

11.5 Benchmarks

The Conflict-Based Stochastic Local Search is benchmarked against two other approaches: a simple thresholding approach and a stochastic MILP.

11.5.1 Rule-based Demand Response

One simple demand response approach is to use thresholds. To benchmark, we test thresholds that divide the activities into *critical* and *non-critical* activities. The simple demand response mechanism then results in the following rules:

- *If battery charge* $\leq x_1$*, shed non-critical activities currently active*
- *If battery charge* $\leq x_2$*, shed critical activities currently active*

where $x_1 > x_2$. While simple, the chief difficulty with rule-based approaches is determining the thresholds, x_1 and x_2. For this work, x_1 and x_2 were set to 0.3 and 0.2 respectively, which provided good empirical performance among thresholds.

11.5.2 Scenario-Based MILP Approach

Because of their widespread use in literature, particularly power systems scheduling, a second benchmark is performed against a stochastic mixed-integer linear program (MILP). Many forms of stochastic MILPs are used and while it is difficult to benchmark against exactly the same formulation due to researchers' various requirements (such as transmission constraints, generation constraints, economic/market constraints), a benchmark is constructed using some of the most salient features.

A challenge in stochastic MILPs is the representation of uncertainty. A popular method is to sample from the relevant distributions and reason over a finite number of scenarios [78]. This presents the challenge of quickly becoming intractable for multistage decision problems and further approximations are often deployed. One approximation is that of lumped demand. While the proposed method derives a policy for each individual activity, many approaches output a policy only for the aggregate resource demand signal [110, 82, 101]. This is a reasonable approximation for large grids but two issues may arise: because the algorithm is unable to reason about individual loads, it is unable to reason separately about critical loads. Many microgrids operate with various critical loads such as lights or cell towers [4]. To compare the MILP against the chance-constrained model, binary decision variables are used to model servicing critical loads and lump all other non-critical demand into an aggregate signal.

The program used to optimize the lumped demand stochastic MILP model is:

$$\min \quad \sum_t \sum_s \ell_s [t] \, P(s) \tag{11.31}$$

$$\begin{aligned} \text{s.t.} \quad & b_s[t+1] = b_s[t] + g_s[t] - \sum_{\mathcal{A}_{crit}} d_{crit,a,s}[t] \\ & - d_s[t] + \ell_s[t] \end{aligned} \tag{11.32}$$

$$0 \leq b_s[t] \leq b_{max} \tag{11.33}$$

$$0 \leq g_s[t] \leq G_s[t] \tag{11.34}$$

$$\sum_s y_{crit,a,s} P(s) \geq 1 - \alpha, \quad y_{crit,a,s} \in \{0,1\} \tag{11.35}$$

$$\begin{aligned} & \bar{p}_a y_{crit,a,s} - d_{crit,a,t} = 0 \\ & \forall \;\; a, t_{dur,min} \leq t \leq t_{dur,max} \end{aligned} \tag{11.36}$$

The objective (line 11.31) seeks to minimize the aggregated expected loss of load. In (11.31), ℓ_t represents the total loss of load at time step t and $P(s)$ is the probability of scenario s. Constraint (11.32) ensures energy is balanced across time steps: $b_s[t]$ is the battery state of charge at time t in scenario s, $g_s[t]$ is the consumed generation, $d_s[t]$ is the lumped demand, and $d_{crit,a,s}[t]$ the power consumed by critical activity a. Constraint (11.33) ensures the battery remains within its capacity and (11.34) allows the model to waste unused generation: $G_s[t]$ is the actual generation at time step t.

Constraints (11.35, 11.36) relate to the chance constraint on critical activities. Variable $y_{crit,a,s}$ indicates whether critical activity a has started in scenario s. Line 11.35 ensures that the critical activity occurs in at least $1-\alpha$ scenarios. Line 11.36 ensures that if a critical activity turns on, it consumes power (denoted by variable $d_{crit,a,t}$) for all time steps in its required duration. In line 11.36, $\bar{p}_a$ is a constant describing activity a's power demand. To mirror the MDP implementation, formulation is changed slightly and includes satisfaction of the chance constraint as a penalty

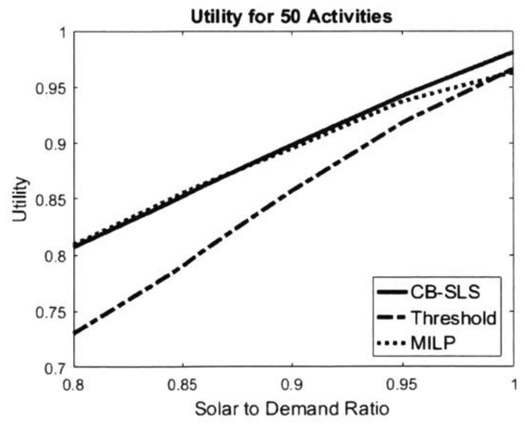


Figure 11-4: Utility as a function of the solar to demand ratio for all three approaches for the 50 activity test cases

Type	St_{lb}	St_{ub}	D_{ub}	P (W)	Crit.
"Fan"	14	17	2	12	No
"Cell Charging"	6	12	3	5	No
"Lighting"	18	20	3	10	Yes

Table 11.1: Example activities used in the tests

term in the objective:

$$\min \quad \sum_t \sum_s \ell_s [t] P(s) + \gamma \sum_i Y_a \tag{11.37}$$

$$\sum_s y_{crit,a,s} P(s) - (1-\alpha) + M Y_a \geq 0, \; Y_a \in \{0,1\} \tag{11.38}$$

where Y_a is a variable that indicates whether the chance constraint for activity a is satisfied. Constraint 11.38 replaces 11.35 and uses big-M notation to ensure that $Y_a = 1$ if the chance constraint is not met.

Regarding modeling the stochastic generation and demand scenarios s, there is a trade off between the MILP's accuracy and complexity. Similar to [17], three scenarios are used denoting times of low, medium, and high demand. To reduce the number of total scenarios, the scenario tree branches at 6 hour intervals for 4 levels over a 24 hour period. Even this approach leads to a scenario tree of size $3^4 = 81$.

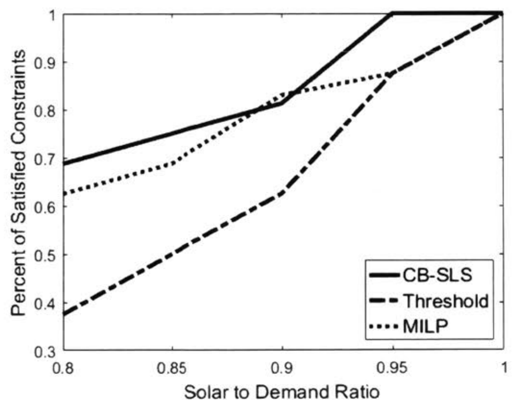


Figure 11-5: Percent of satisfied chance constraints as a function of the solar to demand ratio for the 50 activity test cases

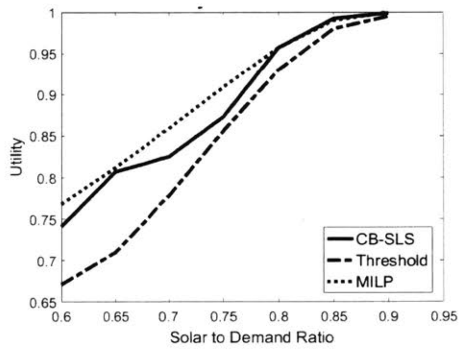


Figure 11-6: Utility as a function of the solar to demand ratio for the 110 activity test cases

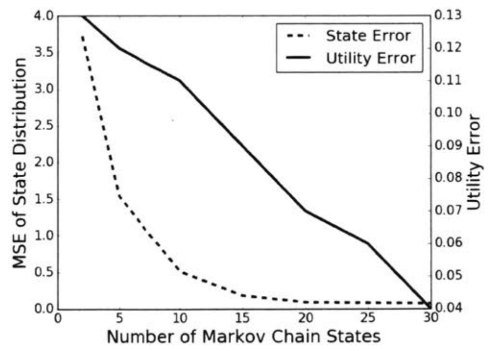


Figure 11-7: Mean squared distribution error and the utility error in approximating the full stochastic scenarios by a Markov chain as a function of the number of Markov chain states

Panel Size (W)	Battery (Ahr)	Chance Constr.
250	200	0.90

Table 11.2: Base generation, storage, and chance constraint parameters for the benchmark tests

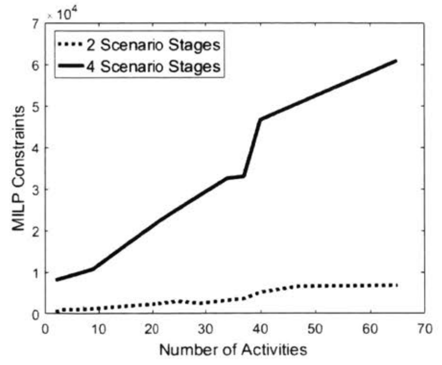


Figure 11-8: Constraints required for 2 and 4 stage MILPs as a function of the number of activities

Case	CB-SLS	Rule-based	MILP
Test Case A	0.64	0.51	0.75
Test Case A C.C.	0.96	0.86	1.0
Test Case B	0.91	0.87	-
Test Case B C.C.	0.83	0.71	-

Table 11.3: Simulation utility of CB-SLS against other control strategies

11.6 Results

The Conflict-Based Stochastic Local Search (CB-SLS) approach is benchmarked against the threshold and MILP approaches. Test cases of various complexity and microgrid over/undersupply were performed. Example electrical activities for the test cases are shown in Table 11.1, which are typical loads for the developing world DC microgrid case. The desired start times and durations were randomized across tests. Lighting activities were marked as critical; each lighting activity was required to receive a 2 hour minimum duration 90% of the time. Table 11.2 shows other problem parameters such as the solar panel and battery size. The generation source's distribution was generated from the output of a 250 W solar panel, using sensor data collected near Jamshedpur, India.

The results of select tests are shown in Figures 11-4, 11-5, 11-6, and 11-7. For the MILP and CB-SLS algorithms, a policy was derived using the respective approach and then this policy was simulated over a large number of days. All reported utilities are from simulations of the output policies. Figure 11-4 was generated by running tests for various cases of over/undersupply for a 50 activity microgrid. Each plotted point represents a different test (i.e. for a test where the average solar to demand ratio is 0.8, the utility for the CB-SLS algorithm is approximately 0.81). A higher solar to demand ratio represents a more oversupplied grid and an easier problem. From 11-4, it is evident that the CB-SLS approach is close to that of the MILP over a wide range of grid conditions. Both approaches outperform the threshold approach. Figure 11-5 compares the algorithms' abilities to satisfy the chance constraints for a wide variety of grid conditions. Once again, the MILP and CB-SLS approaches outperform the thresholding approach with the CB-SLS showing slight improvement over the MILP approach. A reason for this could be the fact that the MILP models only single start times for critical activities; modeling multiple possible start times would require disjunctive constraints and make the model considerably more complex. Fig. 11-6 presents the algorithm scaling to a grid with 110 activities. Results are similar to those of the 50 activity test cases, with CB-SLS performing closely to the MILP.

The CB-SLS relies on discretizing the continuous state space and it is important to consider the accuracy of this discretization. Fig. 11-7 presents the mean squared error between our method's Markov chain distribution and the actual simulated distribution. As the number of states becomes large, the error approaches zero. This presents the trade-off between accuracy and memory complexity: as more modeling accuracy is required, more memory is necessary to reason over more complex scenarios.

11.6.1 Memory Complexity

One of this work's main design constraints is that it should be implemented on low cost microcontrollers for developing world electrification. The state transition matrix is the most memory-intensive element. This has memory complexity $O\left(T * S^2\right)$ where T is the length of the time horizon and S is the number of discrete states at each time step. For a typical case, the time horizon has 24 steps and 9 states are used at each step for a complexity of approximately 2,000 floating point numbers. This yields approximately 8 kilobytes (4 bytes per float), which is feasible on a \$2 microcontroller with approximately 500 kb memory. This contrasts with the MILP approach. Fig. 11-8 plots the number of constraints required for example test cases modeled using 2 and 4 stage scenario trees. The 4 stage MILP requires tens of thousands of constraints for a modest number of activities. Solution of this MILP using a cutting-plane method and simplex tableau costs many megabytes of memory. While both the CB-SLS and MILP approach offer similar performance, the CB-SLS's compact representation of stochasticity allows it to be more easily implemented on embedded systems.

11.7 Conclusion

This chapter presented a lightweight, user-centric stochastic resource allocation algorithm. The algorithm is able to reason directly over distributions of demand and generation and avoid an exponential increase in complexity by modeling the stochastic state through a discrete time and space Markov chain. Despite the state discretization, the approach is able to approximate the continuous state well and compares well against a simple thresholding method and stochastic MILP. Further improvements can be made in how the discretization is constructed as certain discrete states contain little probability mass. Additional improvements lie not only on the algorithmic side, but also in its integration with end users.

Chapter 12

Conclusion

Over the coming years, autonomous systems and robots are rapidly moving from the lab to real world applications. From transportation to manufacturing to elder care, autonomous systems can help humans do tasks more efficiently, reliably, and with lower cost. However, the real world is vastly more complex and uncertain than a lab bench top. When determining how to act, autonomous systems must be able to model uncertainty and reason about it when planning.

This thesis investigated methods for planning under uncertainty in resource constrained systems. One of the challenges of dealing with uncertainty is that it often involves large amounts of computation; the autonomous system must reason about a large number of scenarios that describe what may happen. In planning under uncertainty, many systems are resource-constrained in some way. Autonomous agents and robots that must navigate through their environments are often time-constrained. They must rapidly compute control trajectories in the face of oncoming obstacles. For smart energy systems where lowering cost is a top priority, the systems are constrained in the amount of computational resources to which they have access.

For planning under uncertainty in resource-constrained systems, this thesis makes a number of contributions. For the case of motion planning under uncertainty, this thesis proposes:

- A method to bound the expected cost of a path and integrate the precision with which the stochastic model is evaluated with the search routine

- The Fast Obstacle eXploration (FOX) algorithm that enables complex obstacle geometries to be integrated with a hybrid search
- A method for integrating agents with non-trivial geometry into risk-aware motion planning by transforming a complex integration into a tractable problem in computational geometry
- Shooting Method Monte Carlo, an approach that combines the shooting method of trajectory optimization with Monte Carlo simulation and enables motion planning to be done with respect to nonlinear dynamical systems and non-Gaussian uncertainty

For the path planning domain, extensive test cases were performed using linear models, a Dubins car model, and a Slocum glider model. In the case of resource allocation for smart energy systems, this thesis discussed two approaches:

- A Market for Reliability algorithm where agents trade reliability for a resource and a memory-efficient Monte Carlo simulation determines if the market clears
- A centralized resource allocation algorithm that models state uncertainty using an MDP and a stochastic local search to determine good policies

12.1 Areas of Possible Future Work

In spite of these contributions, there is still a wide range of possible future work and improvements.

12.1.1 Contingency Planning

One of the drawbacks of the motion planning algorithms presented in this thesis is that they focus on computing open loop trajectories. To execute this trajectory, a real world robot would need some form of a closed loop controller or control policy.

It would be insightful to integrate either approach with the algorithms presented in this chapter. A closed loop controller will attempt to return the agent to the open loop trajectory if it deviates off course. While a wide range of algorithms from control theory exist to do this, it would be insightful to also model the uncertainty associated with this closed loop controller.

On the other hand, if certain events transpire, it may be advisable to forget the open loop trajectory and either re-plan or use a policy. A policy-based approach would associate different actions depending on the outcome of the open loop execution.

12.1.2 Determining Initial Guesses

A large drawback of using nonlinear trajectory optimization-based approaches is that they require reasonable initial guesses for the decision variables. These must be determined using human insight, which is often a non-trivial task for complex dynamical systems. An advanced approach could automate this process, perhaps by using learning.

12.1.3 Integration with Activity Planning

The motion planning problems in this thesis focused on planning from an initial state to a final state. Rather than require only a single agent to navigate to a single goal, many real world applications call for multiple agents to accomplish a set of goals. One of the advantages to the hybrid search is that the upper level region planner relies on state space search. Many techniques have been proposed for activity-planning using some form of state space search. It would be very valuable to integrate many of these approaches with the motion planning algorithms presented in this thesis.

12.1.4 Discretizing States for Resource Allocation

In the resource allocation use case, the centralized planner relies on a discretization to model state uncertainty. During planning, many of these states had very low probability density and the density could be concentrated in a small number of states.

This causes the algorithm to waste computational effort on low probability states. A better model would adjust the discretization based on the likelihood of states.

Appendix A

Slocum Glider Dynamics

A model of the glider's dynamics must take into account the changes in linear and rotational momentum. This thesis closely follows the glider model in [70]. This work derives the equations of motion by analyzing the glider's kinetic energy and momentum. The chief difference between the model used in this thesis and in [70] are the control inputs. The model in [70] assumes the glider has access to two control inputs: a stationary point mass whose mass value can be varied and a movable point mass. The stationary point mass models the buoyancy engine; the glider's overall buoyancy is altered by changing its mass. Moving the movable point mass affects the center of buoyancy and consequently the glider's pitch. To simplify the model in this thesis, I assume the glider has a variable mass but uses adjustable fins to change pitch and yaw rather than a movable mass. This removes the movable point mass's state from the equations of motion and allows for a slightly simpler model. An overview of the glider model's geometry and coordinate system is shown in Fig. 7-3.

The governing equations of motion, Eq. 18 in [70] are:

$$\dot{R} = R\hat{\Omega} \tag{A.1}$$

$$\dot{b} = Rv \tag{A.2}$$

$$\dot{\Omega} = J^{-1}T \tag{A.3}$$

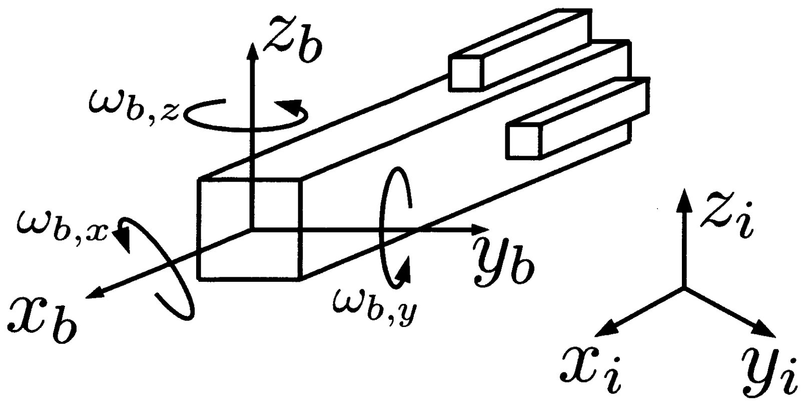


Figure A-1: Overview of the polyhedron finned Slocum glider model used in this thesis. Coordinates (x_i, y_i, z_i) represent inertial coordinates, (x_b, y_b, z_b) represent body coordinates, and angular velocities $(\omega_{b,x}, \omega_{b,y}, \omega_{b,z})$ represent rotations around the respective body coordinates.

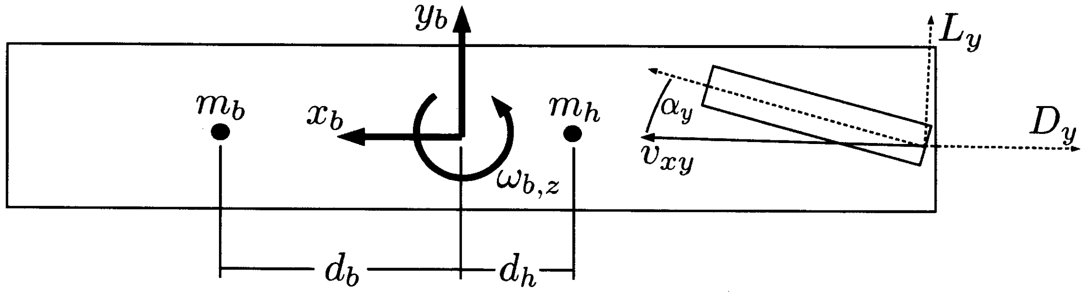


Figure A-2: View of glider forces and geometry in the xy-plane. The xz-plane is similar.

$$\dot{v} = M^{-1}F \tag{A.4}$$

$$\dot{m}_b = u_{m_{buoyancy}} \tag{A.5}$$

where R is the rotation matrix, Ω is the glider's angular velocity, b is the glider's position vector, v is the glider's velocity vector, J is the inertia matrix, M is the mass matrix, m_b is the mass of the buoyancy engine, and u_3 is the control input that represents the change in the buoyancy engine's mass. T is a torque term where:

$$T = J\Omega \times \Omega + Mv \times v + T_{ext} \tag{A.6}$$

and F is a force term with $F = Mv \times \Omega + F_{ext}$. T_{ext} is an external torque term with $T_{ext} = R^T \sum (x_i - b) \times f_{ext,i} + R^T \sum \tau_{ext,j}$ and F_{ext} is an external force term with $F_{ext} = R^T \sum f_{ext,i}$. The reader is referred to [70] for the full details and derivations of each term.

While [70] focuses on control of a 2D test case, this thesis uses a full 3D dynamical model as a test case. The model is derived from the governing Eqs. (A.1) - (A.5). It relies on thirteen states and three control inputs. The states are

$$[x_i, y_i, z_i, v_{x,b}, v_{y,b}, v_{z,b}, \psi, \theta, \phi, \omega_x, \omega_y, \omega_z, m_b] \tag{A.7}$$

States x_i, y_i, z_i are the glider's position states in the inertial reference frame, while states $v_{x,b}, v_{y,b}, v_{z,b}$ are velocity states in the glider's body reference frame. Angles ψ, θ, ϕ are Euler angles that transform the inertial frame to the body frame by a rotation ϕ about the original z axis, a rotation θ about the new y axis, and finally a rotation ψ about the new x axis as described in [106]. States $\omega_x, \omega_y, \omega_z$ are the rates of rotation around the x, y, and z axes in the body frame. m_b is the mass of the buoyancy engine. I assume three control inputs: $u_{pitch}, u_{yaw}, u_{m_{buoyancy}}$. Inputs u_{pitch} and u_{yaw} model input fin angles; $u_{m_{buoyancy}}$ is the change in mass of the buoyancy engine.

In order to model and optimize the glider's trajectory, it is necessary to write the equations of motion in the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ where $\mathbf{x}$ represents a vector of the states and $\mathbf{u}$ a vector of the control inputs. The rate of change of the glider's position in inertial coordinates is provided by transforming the velocity from body to inertial coordinates:

$$\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{bmatrix} = R^T \begin{bmatrix} v_{x,b} \\ v_{y,b} \\ v_{z,b} \end{bmatrix} \tag{A.8}$$

R is the rotation matrix:

$$R = \begin{bmatrix} c\theta c\phi & c\theta s\phi & -s\theta \\ -c\psi s\phi + s\psi s\theta c\phi & c\psi c\phi + s\psi s\theta s\phi & s\psi c\theta \\ s\psi s\phi + c\psi s\theta c\phi & -s\psi c\phi + c\psi s\theta s\phi & c\psi c\theta \end{bmatrix} \tag{A.9}$$

where c and s are abbreviations for *sine* and *cosine.*

The time derivatives of ψ, θ, ϕ are related to $\omega_x, \omega_y, \omega_z$; however, a transformation is required because ω_x , ω_y, and ω_z are with respect to the body coordinates. The two rates of rotation can be related by considering differential rotations as described in [106]. Specifically,

$$\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} 1 & s\psi t\theta & c\psi t\theta \\ 0 & c\psi & -s\psi \\ 0 & s\psi / c\theta & c\psi / c\theta \end{bmatrix} \begin{bmatrix} \omega_{x,b} \\ \omega_{y,b} \\ \omega_{z,b} \end{bmatrix} \tag{A.10}$$

Computing the time derivatives of the glider's velocity and angular velocity terms requires equations (A.4) and (A.3). The time change in body velocity can be written:

$$\dot{v}_{x,body} = 1/m_x \left(F_{coriolis,x} + F_{gravity,hull,x} + F_{gravity,buoyancy,x} + F_{lift,x}\right) \tag{A.11}$$

$$\dot{v}_{y,body} = 1/m_y \left(F_{coriolis,y} + F_{gravity,hull,y} + F_{gravity,buoyancy,y} + F_{lift,y}\right) \tag{A.12}$$

$$\dot{v}_{z,body} = 1/m_z \left(F_{coriolis,z} + F_{gravity,hull,z} + F_{gravity,buoyancy,z} + F_{lift,z}\right) \tag{A.13}$$

The Coriolis forces can be determined from:

$$\mathbf{F}_{coriolis} = \mathbf{M}\mathbf{v}_b \times \mathbf{\Omega} \tag{A.14}$$

$\mathbf{F}_{gravity,hull}$ is simply the gravity force transformed into the body frame of reference:

$$\mathbf{F}_{gravity,hull} = R^T \begin{bmatrix} 0 \\ -g * m_{hull} \\ 0 \end{bmatrix} \tag{A.15}$$

Similarly, $\mathbf{F}_{gravity,buoyancy}$ is the force due to changes in the mass of the buoyancy engine transformed into the body frame of reference:

$$\mathbf{F}_{gravity,buoyancy} = R^T \begin{bmatrix} 0 \\ -g * m_b \\ 0 \end{bmatrix} \tag{A.16}$$

The lift terms, $\mathbf{F}_{lift}$ are the forces due to the glider's fins. $\mathbf{F}_{lift}$ includes lift and drag forces; lift forces are perpendicular to the fin while drag forces are parallel to the fin opposing the fin's motion. In the model used for this thesis, $\mathbf{F}_{lift}$ can be written:

$$F_{lift,x} = L_{pitch} \sin(\alpha_p) - D_{pitch} \cos(\alpha_p) + L_{yaw} \sin(\alpha_y) - D_{yaw} \cos(\alpha_y) \tag{A.17}$$

$$F_{lift,y} = L_{yaw} \cos(\alpha_y) + D_{yaw} \sin(\alpha_y) \tag{A.18}$$

$$F_{lift,z} = - L_{pitch} \cos(\alpha_p) - D_{pitch} \sin(\alpha_p) \tag{A.19}$$

where L and D represent the lift and drag and α_p and α_y represent the angles of attack of the pitch and yaw fins respectively. Often used to compute the lift and drag on airplane wings, the angle of attack is the angle between the approaching fluid velocity and the chord of the fin. The lift and drag of each fin are functions of the angles of attack, the fin speed, and a number of constants. They are described by the equations:

$$L_{pitch} = K_L \alpha_p v_{xz}^2 \tag{A.20}$$

$$D_{pitch} = \left(K_{D,0} + K_D \alpha_p^2\right) v_{xz}^2 \tag{A.21}$$

$$L_{yaw} = K_L \alpha_y v_{xy}^2 \tag{A.22}$$

$$D_{yaw} = \left(K_{D,0} + K_D \alpha_y^2\right) v_{xy}^2 \tag{A.23}$$

Terms v_{xz} and v_{xy} represent the fluid speed at the fin in the xz and xy planes. K_L, $K_{D,0}$, and K_D are lift and drag coefficients. For this thesis, I assumed the values presented in [70]. Variables α_p and α_y are the pitch and yaw angles of attack. The angles of attack can be computed by considering the velocity components at the fins. To model the effect of changing the fin angle on the angle of attack, I assume that the control inputs u_{pitch} and u_{yaw} have an additive effect on the angle of attack:

$$\alpha_p = \text{atan2}\left(v_{pitch,z}, v_{pitch,x}\right) + u_{pitch} \tag{A.24}$$

$$\alpha_y = \text{atan2}\left(v_{yaw,y}, v_{yaw,x}\right) + u_{yaw} \tag{A.25}$$

$$\tag{A.26}$$

where

$$v_{pitch,x} = v_{b,x} \tag{A.27}$$

$$v_{yaw,x} = v_{b,x} \tag{A.28}$$

$$v_{pitch,z} = v_{b,z} + d_b\omega_y \tag{A.29}$$

$$v_{yaw,y} = v_{b,y} - d_b\omega_z \tag{A.30}$$

The change in angular velocities results from Eqs. (A.3) and (A.6):

$$\boldsymbol{\omega} = \boldsymbol{J}^{-1}\left(\boldsymbol{J\Omega} \times \boldsymbol{\Omega} + \boldsymbol{M}\boldsymbol{v} \times \boldsymbol{v} + T_{ext}\right) \tag{A.31}$$

The first two terms on the right hand side of Eq. (A.31) are straightforward to compute given $\boldsymbol{\Omega} = [\omega_{x,b}, \omega_{y,b}, \omega_{z,b}]$ and $\mathbf{v} = [v_{x,b}, v_{y,b}, v_{z,b}]$. T_{ext} can be split into torques due to the fins or due to the force from the buoyancy engine:

$$T_{ext,x} = 0 \tag{A.32}$$

$$T_{ext,y} = T_{fin,pitch} + T_{buoyancy,pitch} \tag{A.33}$$

$$T_{ext,z} = T_{fin,yaw} + T_{buoyancy,yaw} \tag{A.34}$$

where

$$T_{fin,pitch} = - d_f \left(D_{pitch} \sin\left(\alpha_p\right) + L_{pitch} \cos\left(\alpha_p\right)\right) \tag{A.35}$$

$$T_{fin,yaw} = - d_f \left(D_{yaw} \sin\left(\alpha_y\right) + L_{yaw} \cos\left(\alpha_y\right)\right) \tag{A.36}$$

$$T_{buoyancy,pitch} = d_b F_{gravity,buoyancy,z} \tag{A.37}$$

$$T_{buoyancy,yaw} = d_b F_{gravity,buoyancy,y} \tag{A.38}$$

and d_f is the distance from the center of mass to the fin, and d_b is the distance from the center of mass to the buoyancy engine. These distances are shown for the xy-plane in Fig. A-2.

Bibliography

[1] International Energy Agency. *World Energy Outlook.* International Energy Agency, Web, 2013.

[2] Moody Alam, Alex Rogers, and Sarvapali D Ramchurn. Interdependent multi-issue negotiation for energy exchange in remote communities. In *Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, 2013.

[3] Pierre Alliez, Stéphane Tayeb, and Camille Wormser. 3D fast intersection and distance computation. In *CGAL User and Reference Manual.* CGAL Editorial Board, 4.11.1 edition, 2018.

[4] Pär Almqvist and Sarraju Narasinga Rao. The quest to bring power, everywhere in India, December 2015. [Online; posted 09-December-2015].

[5] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 688–694 vol.1, 1999.

[6] Howard Anton. *Calculus: A New Horizon.* John Wiley & Sons, 1999.

[7] Vijay S. Bawa. On chance constrained programming problems with joint constraints. *Management Science*, 19(11):1326–1331, 1973.

[8] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[9] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[10] Subhrajit Bhattacharya. *Topological and Geometric Techniques in Graph-Search Based Robot Planning.* PhD thesis, University of Pennsylvania, January 2012.

[11] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3d. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[12] L. Blackmore. Robust path planning and feedback design under stochastic uncertainty. In *American Institute of Aeronautics and Astronautics Guidance, Navigation and Control*, 2008.

[13] Lars Blackmore, Hui Li, and B. S. Williams. A probabilistic approach to optimal robust path planning with obstacles. *2006 American Control Conference*, pages 7 pp.–, 2006.

[14] Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C. Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE Trans. Robotics*, 26(3):502–517, 2010.

[15] Lars Blackmore, Masahiro Ono, and Brian C. Williams. Chance-constrained optimal path planning with obstacles. *IEEE Trans. Robotics*, 27(6):1080–1094, 2011.

[16] F. Bouffard, F.D. Galiana, and AJ. Conejo. Market-clearing with stochastic security-part i: formulation. *Power Systems, IEEE Transactions on*, 20(4):1818–1826, Nov 2005.

[17] François Bouffard and Francisco D. Galiana. Stochastic security for operations planning with significant wind power generation. *IEEE Transactions on Power Systems*, 23(2):306, 2008.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, 2004.

[19] Stephen Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley. Notes on decomposition methods. April 2008.

[20] Stephen Bradley, Arnoldo Hax, and Thomas Magnanti. *Applied Mathematical Programming.* Addison-Wesley Publishing Company, 1977.

[21] Zdzislaw Brzezniak and Tomasz Zastawniak. *Basic Stochastic Processes.* Springer-Verlag London, London, 1999.

[22] John Burkardt. The truncated normal distribution. 2014.

[23] Richard Camilli, Paraskevi Nomikou, Javier Escartin, Pere Ridao, Angelos Mallios, Stephanos Kilias, and Ariadne Argyraki. The kallisti limnes, carbon dioxide-accumulating subsea pools. *Scientific Reports*, 5, 07 2015.

[24] Andy Campanella. An analysis of the viability and competitiveness of dc microgrids in northern india. Master's thesis, Massachusetts Institute of Technology, 2013.

[25] J.A. Cobano, R. Conde, D. Alejo, and A. Ollero. Path planning based on genetic algorithms and the Monte-Carlo method to avoid aerial vehicle collisions under uncertainties. In *IEEE International Conference on Robotics and Automation, ICRA 2011, May, 2011, Shanghai, China, USA*, pages 4429–4434, 2011.

[26] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org, 2017.

[27] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[28] Michael L. Cui, Daniel D. Harabor, and Alban Grastien. Compromise-free pathfinding on a navigation mesh. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 496–502. AAAI Press, 2017.

[29] Russ Davis, Charles C. Eriksen, and Clayton P. Jones. Autonomous buoyancy-driven underwater gliders. 11 2002.

[30] Robin Deits and Russ Tedrake. *Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming*, pages 109–124. Springer International Publishing, Cham, 2015.

[31] Devdatt Dubhashi, Fabrizio Grandoni, and Alessandro Panconesi. Distributed approximation algorithms via lp-duality and randomization, 2007.

[32] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.

[33] Christer Ericson. The morgan kaufmann series in interactive 3d technology. In *Real-Time Collision Detection*, The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann, San Francisco, 2005.

[34] Enrique Fernandez and Brian Williams. Mixed discrete-continuous planning with convex optimization. In *AAAI Conference on Artificial Intelligence*, pages 4574–4580, 2017.

[35] Robert M. Freund. Issues in non-convex optimization, April 2004.

[36] Alan Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1(2):141–149, 1992.

[37] Philip E. Gill, Walter Murray, Michael A. Saunders, and Elizabeth Wong. User's guide for SNOPT 7.6: Software for large-scale nonlinear programming. Center for Computational Mathematics Report CCoM 17-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2017.

[38] Enrique Fernandez Gonzalez. *Generative multi-robot task and motion planning over long horizons.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, 1 2018.

[39] Hawai'i Mapping Research Group. Main hawaiian islands multibeam bathymetry and backscatter synthesis. http://www.soest.hawaii.edu/hmrg/multibeam/index.php.

[40] Philippe Guigue and Olivier Devillers. Fast and robust triangle-triangle overlap test using orientation predicates. *J. Graphics, GPU, & Game Tools*, 8(1):25–32, 2003.

[41] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[42] Kris Hauser. Learning the problem-optimum map: Analysis and application to global optimization in robotics. volume abs/1605.04636, 2016.

[43] Hawai'i Mapping Research Group. Main Hawaiian Islands Multibeam Bathymetry and Backscatter Synthesis, 2014.

[44] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[45] Michael Hoy, Alexey S. Matveev, and Andrey V. Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, 33(3):463–497, 2015.

[46] S. Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 807–814, Sep. 2008.

[47] Leonid Hurwicz. The design of mechanisms for resource allocation. *The American Economic Review*, 63(2):1–30, May 1973.

[48] Brian Ichter, Edward Schmerling, Ali-akbar Agha-mohammadi, and Marco Pavone. Real-time stochastic kinodynamic motion planning via multiobjective search on GPUs. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore*, pages 5019–5026, 2017.

[49] Wardah Inam, Daniel Strawser, Khurran Afridi, Rajeev Ram, and David Perreault. Architecture and system analysis of microgrids with peer-to-peer electricity sharing to create a marketplace which enables energy access. *9th International Conference on Power Electronics*, June 2015.

[50] IRENA. Renewable power generation costs in 2017. 2018.

[51] Shweta Jain, Narayanaswamy Balakrishnan, Yadati Narahari, Saiful A. Hussain, and Nyuk Yoong Voo. Constrained tatonnement for fast and incentive compatible distributed demand management in smart grids. In *Proceedings of the Fourth International Conference on Future Energy Systems*, e-Energy '13, pages 125–136, New York, NY, USA, 2013. ACM.

[52] Lucas Janson, Edward Schmerling, and Marco Pavone. *Monte Carlo Motion Planning for Robot Trajectory Optimization Under Uncertainty*, pages 343–361. Springer International Publishing, Cham, 2018.

[53] N. Jetchev and M. Toussaint. Trajectory prediction in cluttered voxel environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 2523–2528, May 2010.

[54] P. Jimenez, F. Thomas, and C. Torras. Collision detection algorithms for motion planning. In Jean-Paul Laumond, editor, *Robot Motion Planning and Control*, chapter 6, pages 305–343. Springer, 1998.

[55] Angus Johnson. Clipper - an open source freeware library for clipping and offsetting lines and polygons., 2010-2014.

[56] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011.

[57] T. Kato, H. Takahashi, K. Sasai, Yujin Lim, Hak-Man Kim, G. Kitagata, and T. Kinoshita. Multiagent system for priority-based load shedding in microgrid. In *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, pages 540–545, July 2013.

[58] Lydia E. Kavraki and Steven M. LaValle. *Motion Planning*, pages 109–131. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[59] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59:849–904, 01 2017.

[60] Matthew P. Kelly. Transcription Methods for Trajectory Optimization: a beginners tutorial. *arXiv e-prints*, page arXiv:1707.00284, July 2017.

[61] Lutz Kettner. Halfedge data structures. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition, 2018.

[62] Sujin Kim, Raghu Pasupathy, and Shane Henderson. A guide to sample-average approximation. Technical report, Cornell, Oct. 2011.

[63] Anton J. Kleywegt and Alexander Shapiro. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, page 502, 2001.

[64] P.E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Stochastic Modelling and Applied Probability. Springer Berlin Heidelberg, 2011.

[65] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, 2010.

[66] TrineKrogh Kristoffersen and Stein-Erik Fleten. Stochastic programming models for short-term power generation scheduling and bidding. pages 187–200, 2010.

[67] B. Kroposki, B. Johnson, Y. Zhang, V. Gevorgian, P. Denholm, B. Hodge, and B. Hannegan. Achieving a 100with extremely high levels of variable renewable energy. *IEEE Power and Energy Magazine*, 15(2):61–73, March 2017.

[68] Damián Laloux and Michel Rivier. *Technology and Operation of Electric Power Systems*, pages 1–46. Springer London, London, 2013.

[69] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, New York, NY, USA, 2006.

[70] N. E. Leonard and J. G. Graver. Model-based feedback control of autonomous underwater gliders. *IEEE Journal of Oceanic Engineering*, 26(4):633–645, Oct 2001.

[71] Jeff Linderoth. Bounds in stochastic programming, March 2003.

[72] Stephen Littlechild. Fereidoon p. sioshansi, editor, competitive electricity markets: Design, implementation and performance, elsevier, amsterdam (2008) xxxiv+582pp. isbn: 978-0-08-047172-3. *Energy Policy*, 37(1):375–376, 2009.

[73] Miles Lubin, Daniel Bienstock, and Juan Pablo. Two-sided linear chance constraints and extensions. 2016.

[74] Brandon Luders, Mangal Kothari, and Jonathan P. How. Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Toronto, Canada, August 2010.

[75] Sadhan Mahapatra and S. Dasappa. Rural electrification: Optimising the choice between decentralised renewable energy sources and grid extension. *Energy for Sustainable Development*, 16(2):146 – 154, 2012.

[76] Nicholas Melchior and Reid Simmons. Particle RRT for path planning with uncertainty. In *2007 IEEE International Conference on Robotics and Automation*, pages 1617–1624, Pittsburgh, PA, April 2007.

[77] Taher Niknam, Rasoul Azizipanah-Abarghooee, and Mohammad Rasoul Narimani. An efficient scenario-based stochastic programming framework for multi-objective optimal micro-grid operation. *Applied Energy*, 99(0):455 – 470, 2012.

[78] Matthias P. Nowak and Werner Römisch. Stochastic lagrangian relaxation applied to power scheduling in a hydro-thermal system under uncertainty. *Annals of Operations Research*, 100(1):251–272, 2000.

[79] Masahiro Ono and Brian C. Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Proceedings of the 47th IEEE Conference on Decision and Control, CDC 2008, December 9-11, 2008, Cancún, Mexico*, pages 3427–3432, 2008.

[80] Masahiro Ono, Brian C. Williams, and Lars Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk. *J. Artif. Intell. Res. (JAIR)*, 46:511–577, 2013.

[81] Art B. Owen. *Monte Carlo theory, methods and examples*. Stanford University, 2013.

[82] M. Parvania and M. Fotuhi-Firuzabad. Demand response scheduling by stochastic scuc. *IEEE Transactions on Smart Grid*, 1(1):89–98, June 2010.

[83] Sachin Patil, Jur van den Berg, and Ron Alterovitz. Estimating probability of collision for safe motion planning under Gaussian motion and sensing uncertainty. In *IEEE International Conference on Robotics and Automation, ICRA 2012, St. Paul, Minnesota, USA*, 2012.

[84] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the Conference on Visualization '02*, VIS '02, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[85] Mikhail Pivtoraiko, Ross Alan Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, March 2009.

[86] Gregory Plett. Linear Quadratic Gaussian. 2016.

[87] Devon Powell and Tom Abel. An exact general remeshing scheme applied to physically conservative voxelization. *Journal of Computational Physics*, 297:340 – 356, 2015.

[88] Andras Prékopa. *Stochastic programming*. Kluwer Academic Publishers Group, Dordrecht; Norwell, MA, 1995.

[89] J.H. Reif and University of Rochester. Department of Computer Science. *Complexity of the Mover's Problem and Generalizations*. Reports // ROCHESTER UNIV NY. Department of Computer Science, University of Rochester, 1979.

[90] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[91] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[92] Amin Saberi. Market and game dynamics. 2009.

[93] Hussain A. Samad, Shahidur R. Khandker, M. Asaduzzaman, and Mohammad Yunu. The benefits of solar home systems: An analysis from bangladesh. December 2013.

[94] P. Samadi, H. Mohsenian-Rad, R. Schober, and V.W.S. Wong. Advanced demand side management for the future smart grid using mechanism design. *Smart Grid, IEEE Transactions on*, 3(3):1170–1180, Sept 2012.

[95] Pedro Santana. *Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, 7 2016.

[96] Pedro Santana, Sylvie Thiébaux, and Brian Williams. RAO*: An algorithm for chance-constrained POMDPs. In *AAAI Conference on Artificial Intelligence*, pages 3308–3314, 2016.

[97] Daniel Schnitzer, Deepa Shinde Lounsbury, Juan Pablo Carvallo, Ranjit Deshmukh, Jay Apt, and Daniel M. Kammen. Microgrids for rural electrification: A critical review of best practices based on seven case studies. Technical report, United Nations Foundation, February 2014.

[98] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. volume 33, pages 1251–1270, Thousand Oaks, CA, USA, August 2014. Sage Publications, Inc.

[99] John Shlens. A tutorial on principal component analysis, March 2003.

[100] Daniel Strawser, Brian C. Williams, and Wardah Inam. A market for reliability for electricity scheduling in developing world microgrids. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS 2015. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[101] W. Su, J. Wang, and J. Roh. Stochastic energy scheduling in microgrids with intermittent renewable energy resources. *IEEE Transactions on Smart Grid*, 5(4):1876–1883, July 2014.

[102] Ioan A. Şucan and Lydia E. Kavraki. *Kinodynamic Motion Planning by Interior-Exterior Cell Exploration*, pages 449–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[103] Teledyne Webb Research. *Slocum G2 Glider Operators Manual.*

[104] Selim Temizer, Mykel J. Kochenderfer, Leslie P. Kaelbling, Tomás Lozano-pérez, and James K. Kuchar. Collision avoidance for unmanned aircraft using markov decision processes. In *AIAA Guidance, Navigation, and Control Conference*, Toronto, Canada, 2010.

[105] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 4.13 edition, 2018.

[106] Michael Triantafyllou. Kinematics of moving frames, Fall 2004.

[107] Wayes Tushar, Jian A. Zhang, David B. Smith, H. Vincent Poor, and Sylvie Thiébaux. Prioritizing consumers in smart grid: A game theoretic approach. *CoRR*, abs/1312.0659, 2013.

[108] Jur van den Berg, Pieter Abbeel, and Kenneth Y. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *I. J. Robotic Res.*, 30(7):895–913, 2011.

[109] Menkes van den Briel, Paul Scott, and Sylvie Thiebaux. Randomized load control: A simple distributed approach for scheduling smart appliances. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, China, August 2013.

[110] P. P. Varaiya, F. F. Wu, and J. W. Bialek. Smart operation of smart grid: Risk-limiting dispatch. *Proceedings of the IEEE*, 99(1):40–57, Jan 2011.

[111] Bala R. Vatti. A generic solution to polygon clipping. *Commun. ACM*, 35(7):56–63, July 1992.

[112] Andreas Veit, Ying Xu, Ronghuo Zheng, Nilanjan Chakraborty, and Katia Sycara. Multiagent coordination for energy consumption scheduling in consumer cooperatives. 2013.

[113] Perukrishnen Vytelingum, Sarvapali D. Ramchurn, Thomas D. Voice, Alex Rogers, and Nicholas R. Jennings. Trading agents for the smart electricity grid. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 897–904, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[114] Qi Wang, Chunyu Zhang, Yi Ding, George Xydis, Jianhui Wang, and Jacob Østergaard. Review of real-time electricity markets for integrating distributed energy resources and demand response. *Applied Energy*, 138:695 – 706, 2015.

[115] Douglas C. Webb, Paul J. Simonetti, and Clayton P. Jones. Slocum: An underwater glider propelled by environmental energy. *IEEE Journal of Ocean Engineering*, 26(4):447–452, 2001.

[116] Eric W. Weisstein. Bivariate normal distribution. From MathWorld—A Wolfram Web Resource, 2018. Last visited on 03/3/2018.

[117] Xiong Wu, Xiuli Wang, and Zhaohong Bie. Optimal generation scheduling of a microgrid. In *Innovative Smart Grid Technologies (ISGT Europe), 2012 3rd IEEE PES International Conference and Exhibition on*, pages 1–7, Oct 2012.

[118] Masaki Yo, Masahiro Ono, Brian C. Williams, and Shuichi Adachi. Risk-limiting, market-based power dispatch and pricing. In *European Control Conference*, July 2013.

[119] Jichen Zhang, J.D. Fuller, and S. Elhedhli. A stochastic programming model for a day-ahead electricity market with real-time reserve shortage pricing. *Power Systems, IEEE Transactions on*, 25(2):703–713, May 2010.

[120] J. Ziegler and C. Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1879–1884, Oct 2009.