Fall 2018

Algorithms for Robot Coverage Under Movement and Sensing Constraints

Jeremy S. Lewis

Algorithms for Robot Coverage under Movement and Sensing Constraints

by

Jeremy S Lewis

Bachelor of Science
Francis Marion University 2008

Master of Science
University of South Carolina 2011

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2018

Accepted by:

Jason O'Kane, Major Professor

Ioannis Rekleitis, Committee Member

Marco Valtorta, Committee Member

Bin Zhang, Committee Member

Manton Matthews, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Acknowledgments

First, I would like to thank my advisor, Jason M. O'Kane. Without your support and push, I would not be receiving this degree. When you met me at Queen St. Grocery in Charleston, SC while taking your son to a doctor's appointment, you not only gave me the motivation to get back and finish, but also the spark needed to inspire my imagination to see something more. Thank you, Jason.

I would also like to thank all of my friends that put up with my crazy life on this extended journey I have called a PhD. Without your support, I would not have had those moments of peace and excitement needed to reenergize and keep progressing.

Last, but far from least, I would like to thank Brianna Dennis. She offered me support and encouragement, but more than anything gave me her understanding. Brianna, thank you for understanding where I am and what I have been doing. I doubt I can ever tell you how much I needed it.

Abstract

This thesis explores the problem of generating coverage paths—that is, paths that pass within some sensor footprint of every point in an environment—for mobile robots. It both considers models for which navigation is a solved problem but motions are constrained, as well for models in which navigation must be considered along with coverage planning because of the robot's unreliable sensing and movements.

The motion constraint we adopt for the former is a common constraint, that of a Dubins vehicle. We extend previous work that solves this coverage problem as a traveling salesman problem (TSP) by introducing a practical heuristic algorithm to reduce runtime while maintaining near-optimal path length. Furthermore, we show that generating an optimal coverage path is NP-hard by reducing from the Exact Cover problem, which provides justification for our algorithm's conversion of Dubins coverage instances to TSP instances. Extensive experiments demonstrate that the algorithm does indeed produce path lengths comparable to optimal in significantly less time.

In the second model, we consider the problem of coverage planning for a particular type of very simple mobile robot. The robot must be able to translate in a commanded direction (specified in a global reference frame), with bounded error on the motion direction, until reaching the environment boundary.

The objective, for a given environment map, is to generate a sequence of motions that is guaranteed to cover as large a portion of that environment as possible, in spite of the severe limits on the robot's sensing and actuation abilities.

We show how to model the knowledge available to this kind of robot about its own position within the environment, show how to compute the region whose coverage can be guaranteed for a given plan, and characterize regions whose coverage cannot be guaranteed by any plan. We also describe an algorithm that generates coverage plans for this robot, based on a search across a specially-constructed graph. Simulation results demonstrate the effectiveness of the approach.

Table of Contents

List of Figures

Chapter 1

Introduction

Robots have become an accepted part of life. The U.S. Federal Aviation Administration (FAA) estimates that 5.5 million unmanned aircraft will be sold in 2018, with an increase of 600,000 to 6.1 million units sold in 2019 [40]. The robot vacuum manufacturer iRobot boasted a 34% increase in revenue in 2017 after a year of focused marketing of their home robots [57]. Boston Dynamics has created a collection of robots capable of running, jumping, and carrying loads as a pack-mule with movement schemes ranging from two and four-legged walking robots to legged robots with wheeled feet and blends of wheels and legs [19].

One task to which any of these robots may be placed is coverage. Coverage is the task of visiting every point in a finite space. When every point is visited, the space is covered. Obviously in any space (no matter how small) every point cannot be individually visited, therefore we define coverage as: given a known environment and a robot, pass the robot within a specified range every point in the environment [29, 30, 47]. A solution to the problem can be stated as a path which passes within the specified range of every point in the environment.

1.1 Motivation

The coverage problem has very real-world and in-demand applications. Consider the iRobot Roomba vacuum cleaner which works by passing its cleaning surface over all parts of a floor, as does a robotic lawn mower in a yard. Both of these robots are solving the coverage problem [94]. Similar to both vacuuming and lawn maintenance, humanitarian de-mining operations [87] require the solution to this problem with much more costly consequences for an incorrect or incomplete plan. Robotic farming has also arrived, with some implementations using a manipulator and specially constructed environment to maintain the plants [41]. Though it seems like a very different application, a coverage problem exists because the manipulator must visit different locations in the environment without necessarily knowing ahead of time which areas must be visited.

Another task which can be solved as a coverage problem is surveillance with limited or inhibited sensor range. Ideally, one has enough sensors to passively monitor an environment completely. Some environments are too large to be reasonably monitored in this fashion. In this case, mobile robots carrying the sensors may be deployed to monitor an area too large for the sensors to be arranged within some distance of every point in the environment. The robots move around the environment providing complete and continuous coverage [82]. Search-and-rescue [21] is a very important coverage problem in which lives may be saved. Even painting [11] is an instance of the coverage problem.

To solve a coverage problem, one must also solve the navigation problem. Plainly put, navigation is the task of moving a robot from one point to another with consideration to the means by which a robot translates. Though simply described, navigation problems are anything but simple to solve. Robot navigation problems are made difficult due to the problems of state estimation, physical capabilities (and in-capabilities) of physical devices, and environmental concerns such as dangers to the robot or dangers to the environment by the robot. These challenges aside, there is still the actual problem of developing a plan to move to a point from another through a space which may not allow any direct path. If the path between a robot's location and goal location is not simple, it might require a motion planner.

A navigation problem is stated in terms of a location or series of locations to which a robot must translate via motions. A planner used to generate solutions to robot problems involving navigation is a type of motion planner. When describing a problem using a physical, sensing, actuated device—a robot—the solution may be modeled as a sequence of possible robot motions (or actions). Solutions of this sort are usually generated by open-loop offline planners. The problem may be stated in terms of an input or initial state (or not, in the case of a kidnapped robot with recovery [28, 35]) and a goal state. The solution is a sequence of robot actions which result in the goal state. Solutions exist in one of two forms: probabilistic, in which a solution is given with some likelihood of success and deterministic, in which success is guaranteed within the bounds of the system. In contrast to the planner thusly described, an online closed loop planner would specify an action for any point at which the robot may find itself.

Another possibility is to use both types of planners. A global planner produces an ordered set of points through which the robot must pass. The robot then uses a local planner to generate an online policy which maps its current location and goal location to an action. As in [88], it is not uncommon for there to exist multi-stage planners that combine global and local planners. In this separation of concerns, a global planner may run offline to generate an overall or large scale plan, while a local planner runs online to execute the plan.

In large-scale navigation (and therefore coverage) problems, state estimation is hard because the inaccuracies of actuations accumulate and sensors might not be powerful enough to directly offset positional uncertainty. Planners must either be provided a sensor with some global frame of reference to ensure their prescribed actions result in the correct state or must schedule actions to reduce uncertainty. It is even possible to have a motion planner—or part of a motion planner—dedicated to localization [72, 80, 81].

In addition to the problem of state estimation, environmental complexity and the movement constraints of the covering vehicle can be problematic. Less irregular and obstacle-free environments allow a larger solution space, while environments with obstacles and irregular shapes reduce the number of reasonable coverage plans. Vehicles with holonomic movement constraints can follow paths along any trajectory and so do not limit the solution space. Robots with non-holonomic constraints are restricted in their movement and so restrict the solution space further, possibly resulting in an empty space.

Even when a solution to the problem exists, it might be a terrible solution. It is trivial to say that a robot with a known fixed-area sensor must follow a path of some minimal length to cover an area with that sensor. An optimal solution would be one which is no longer than the theoretical minimum necessary for coverage. A non-holonomic vehicle may not even be able to follow this minimum path and so there may exist a different minimum length for any given non-holonomic vehicle type. Note this is not a pedantic question. A less optimal solution is a more expensive solution in both time and resources; energy, likely, is the most scarce. Given the expansive areas some planners are given to find coverage, a poor enough plan might not even be implementable. At a minimum, execution will need to be halted for refueling.

1.2 Motivation

Optimal plans with holonomic constraints

At the time of this work, the state of the art for generating optimal coverage plans is in the work by Xu, Viriyasuthee, and Rekleitis [107]. The authors present a very fast algorithm constructing a plan for a vehicle to cover a known region. Their algorithm depends on a seed-spreader coverage technique. That is, a back-and-forth linear motion from one environment edge to another. After each motion, the robot makes a motion perpendicular to its direction of coverage, a distance equal to its sensor's radius. They use a cellular decomposition to break their environment into regions, each capable of being covered by a seed-spreader motion plan. The plan is executed by a UAV with GPS-enabled way-point following. The results are shown in Figure 1.1. It is evident that entry and exit from coverage area—the area outside of the shadowed regions—is problematic. The region covered by the sensor creates coverage paths which are too close together which results in a key-hole flight path, at best. The planner is unable to take into account that the distance between successive coverage passes is greater than twice the vehicle's turn radius.

Due to the deconstruction method, their algorithm must also consider large sections of the to-be-covered regions in isolation. One side-effect of the discretization is apparent in upper shadowed region of the map (a) in Figure 1.1. Because the planner must consider regions across the shadowed area in isolation, it cannot choose to cross back-and-forth between them. The large loop which crosses the entirety of the shadowed region's width there cannot, instead, continue on and cover a portion of the area to the right of the shadow.

If the region to be covered is large enough, then covering from one "side" to the other and the translating back might be a costly decision. Ideally, the planner should plan coverage both beginning and ending at the starting point with coverage occurring constantly between. Their algorithm does just this; however, due again to their discretization, undesirable behavior sometimes occurs. Consider the coverage plan above and below the shadow in image (a) or left and right of the shadow in image (b) of Figure 1.1. Those large regions are split perpendicular to the axis of coverage. This allows the authors' algorithm to plan a coverage path "out" from the starting point and then back "in" to the starting point.

The authors present a very fast algorithm for covering a large space with a mobile robot. That speed is due to their discretization of the environment and use of the

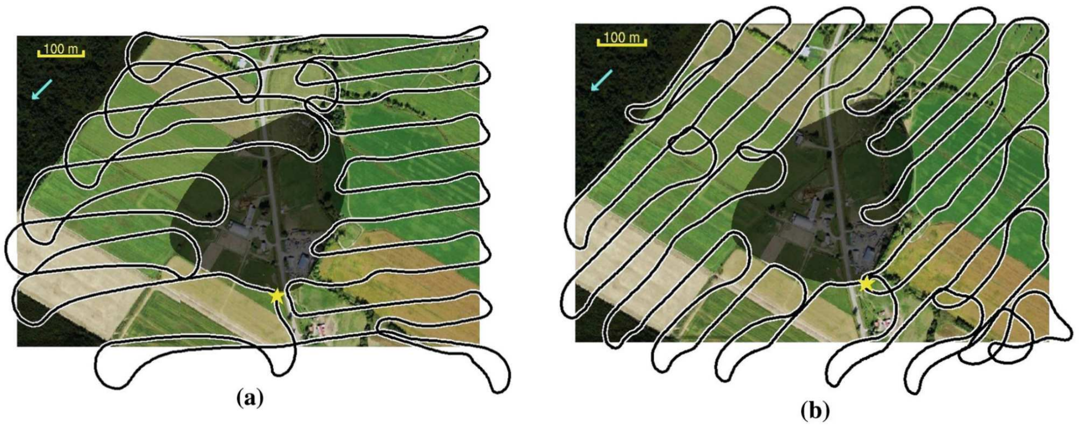

Figure 1.1 A plan created with the method presented by Xu, Viriyasuthee and Rekleitis [107] and executed by a fixed wing UAV.

seed-spreader behavior. But if an optimal path is to be planned, then at a minimum the planner should be able to make global decisions based on the robot's motion constraints. Also, it would need to be aware of the difference between obstacle space—through which translation is impossible or at least very expensive compared to free—and space that is just not interesting to cover. In Section 1.3 we introduce our algorithm which does just that.

Complex robots

As in the work described above [107], when solving coverage problems it is common to assume the navigation problem is solved [67, 102]. To enable this assumption, either a carefully crafted environment must be engineered or a host of sensors are required. Additionally, a local planner is required to execute the plan generated by the global planner along with powerful enough processing to execute that planner before or along with the local planner. This combination means that planning must be done, at least partially, online. While this is acceptable for many problems, the cost of the robot necessary to execute plans may prohibit its application. When the iRobot Roobma presented its solution to the consumer-grade automated vacuum, it used a mostly random algorithm to provide coverage [56]. iRobot increased the robot's sensors to include an IR sensor. This iteration of the robot performs SLAM to solve the coverage problem. These changes increase the cost of the device and introduce more points of failure.

However, if one agrees with Occam and what is wanted or required is a very simple robot model which can make guarantees of coverage, Section 1.4 introduces our exploration of an incredibly simple robot model, which despite error-prone motions can make guarantees on environment coverage. Our algorithm solves both the navigation problem and coverage problem with an offline planner and is able to provide the portion of the environment for which it will certainly cover.

1.3 Dubins Coverage

Our first main contribution is an algorithm that improves upon Xu, Viriyasuthee, and Rekleitis [107] by providing a novel deconstruction method for a known environment which allows for optimal-length paths. Our solution addresses the problem of generating a circuit—the shortest path visiting every necessary point while starting and ending at a given point—by providing a means for a natural shortest path out and back. The planner may skip areas while covering out toward the furthest points from the starting point using them to return while covering if that results in a shorter path.

In addition to considering generating a circuit, the deconstruction also allows a planner to choose the next best step with respect to both distance between steps and movement constraints of the robot. Using this deconstruction, the seed-spreader behavior is not necessarily prescribed, but may be "discovered" by the planner.

Using our deconstruction, in Chapter 3 we prove that calculating the optimal path to cover an environment is an NP-Hard problem. Given that knowledge, we

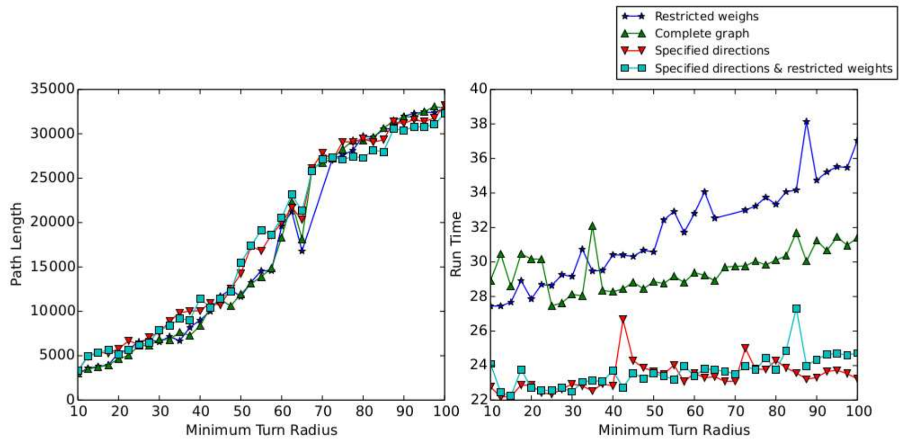

Figure 1.2 The results of our algorithm planning to cover an environment compared to three of our heuristics.

present a family of heuristics to improve the runtime of coverage path calculation. The algorithms are run against several simulated environments and their performance evaluated. Figure 1.2 presents the results of our implementation with the complete graph provided as a baseline along with our heuristics' performance.

In Chapter 4 we address the well-known problem of coverage with a vehicle with non-holonomic movement constraints. We address the problem of large coverage plans and the need to minimize the length of those plans. We select a common movement constraint—the restriction of the minimum turning radius of a vehicle. This is a common restriction in aircraft, water surface, and vehicles using Ackermann steering. To find paths which these vehicles may follow, we consider a planner which must generate actions a Dubins vehicle could execute [33]. We specifically seek sequences of actions a fixed-wing aircraft performing aerial coverage with a camera or a surface water vehicle performing coverage of bodies of water with sonar could execute.

The above scenario is often considered by researchers solving coverage problems due to its immediate experimental and real-world capabilities. The specific robot we imagine executing our plans is a water surface vehicle with a down-facing sonar, similar to the one shown in Figure 1.3 a vehicle used by Kimball, et al. [62]. Our initial

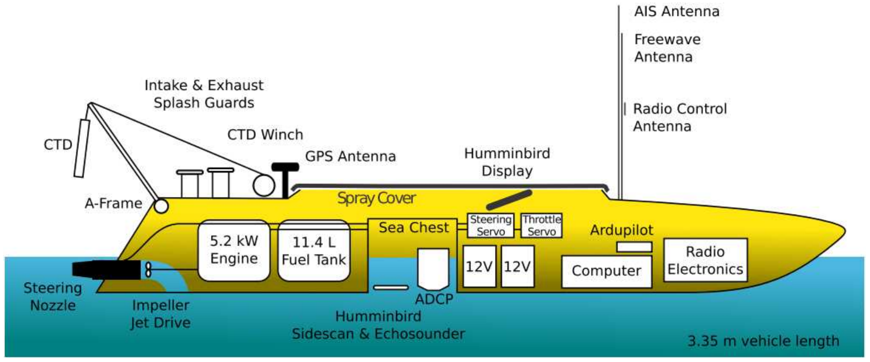

Figure 1.3 A autonomous watercraft with down-facing Hummingbird Echosounder sonar. [62]

application is providing automated coverage for depth maps of the floors of bodies of water. An immediate usage of this research is in maintaining maps of depths of channels used by ocean-going container ships. Other applications might study coral reefs with an additional RGB camera.

We begin with the case of a water surface vehicle, assuming the environment and a portion of its surroundings are sufficiently deep enough to avoid collisions. Aircraft are assumed to be flying high enough to avoid environmental intersection and extend this to problems involving robots performing coverage in water deep enough to avoid collision. We then define the portion of the environment as "don't care" regions. This gives us regions we care about covering and regions about which we do not. This formulation we call Dubins coverage.

One common approach to coverage is using a method known as boustrophedon coverage or "the way of the ox." A vehicle following this path enters the coverage area, continues to the end of the area, performs a rotation of π, re-enters the coverage region, and repeats the process. This process gets its name from the way an ox is used to pull a plow up and down a field. A vehicle using this method requires a "sweep-able" [18] or monotone polygon [86]—that is a polygon monotonic with respect to one axis. The vehicle can complete coverage by simply executing the boustrophedon coverage method as described, along linear sweeps aligned with the axis to which each cell is aligned. There exists an algorithm named for this type of coverage, discretizing an environment into monotonic cells which are well-suited for boustrophedon coverage.

This process is called boustrophedon cellular decomposition (BCD) [4, 25, 30]. Each cell of the decomposition is typically, and independently, covered by a boustrophedon coverage plan. Because the cells are covered independently, most of the decision planning lies in ordering and modifying the cells for coverage. A common means of decomposition is to run a sweep line over the environment, beginning and ending cells at critical points of the environment. After decomposing the environment and sequencing the resulting cells, an algorithm encodes the boustrophedon coverage paths into a coverage plan for the environment. A more in-depth discussion of the method [107] appears in Section 2.

The water surface vehicle is assumed to have complete information about its state and environment. This is accomplished using global positioning sensors as well as a depth camera. Practical implementations using vehicles with Dubins kinematics, such as boats [44] or fixed wing UAVs [85], often do not address the constraint while planning a solution to the coverage problem, leaving the low-level controller to handle the trajectory generation. Our planner builds plans to specifically address the movement constraints of the vehicle while still leaving a local planner to handle following the path. A rigorous definition of our problem appears in Chapter 3.1.

Though BCD with boustrophedon coverage leads to very fast solutions and online-capable planners, there are some downsides: the planner cannot consider the way in which each cell is covered in context of the larger problem, cannot consider the turning constraints of the robots executing them, and may have to modify the cells. In this dissertation takes a different approach—we blend the coverage of individual cells with the ordering and sequencing in an attempt to paths which provide coverage plans considering the movement constraints of the environment.

Minimal decomposition

Our goal is to select tuples specified by the point at which the action should be executed, the action given as a direction (speed is constant), and a duration to execute the action, representing some "atomic" unit of coverage at a specified heading. In the extreme case, one might consider the coverage provided by every action starting from every point in the environment of every duration. This would result in quite a few actions; an infinite set of actions to be exact. To make the number finite, our decomposition in Section 4.1 leaves us with a set of objects representing the minimal amount of coverage which would be accomplished with a prescribed linear pass over the environment along some axis (as defined in Section 3.1). The starting point is specified and rather than storing duration, we store the resulting end point of the action.

We begin with a line sweep BCD to generate the cells which are known to be coverable by a boustrophedon path. We then further split the cells into passes along the same axis as the sweep line generating the cells. The splits should be of width no greater than the diameter of the robot's coverage sensor. The process results in a set of passes which, if all covered mean that the environment is covered. If the cells are decomposed into these passes, an obvious choice is to create a graph with nodes representing the passes and edges representing the distance between the nodes. It is also obvious at this point that coverage requires a path through the environment visiting each cell exactly once. This is, of course, an instance of the Traveling Salesman Problem (TSP). To achieve optimality, however, one must consider that a pass or vertex could be visited in one of two directions along the given axis. In this case, there must actually be two nodes in the graph—one representing the pass covered in each direction. We note that both nodes need not be visited. Covering the pass in one direction is as good as the other. In this case, we group each pair of nodes into a set and the problem generalizes to a special instance of the TSP called set TSP (sTSP) or General TSP (GTSP).

Proof of hardness

Solving a TSP as part of a planner may seem like an overestimation of the complexity of the coverage problem, however we prove in Chapter 3 that Dubins coverage is an NP-complete problem. Our proof draws heavily from Padimitriou's proof of the Hardness of the Euclidean traveling salesman problem [83] and Savla, Frazzoli, and Bullo's work on the hardness of the Dubins traveling salesman problem [96] to perform a reduction from the Exact Coverage problem.

Heuristic

This is bad news for optimal coverage plans which consider the robot's movement constraints. The good news is that because the runtime of TSP experiences non-polynomial growth in the number of nodes and edges. In Chapter 4.2 we devise and compare several heuristics by which we reduce the number of one or both. Though optimality is marginally reduced, we achieve much better run times.

In a general sense, a heuristic's goal is to reduce choices which are not likely to lead to an optimal solution, while preserving many of the choices which do. Our first and most aggressive attempt simply specifies the direction which selects which of the two nodes in each set is kept—it drops half of the nodes and connected edges in the graph. The other three are less aggressive, selecting edges which are less likely to be included in a minimal circuit.

We present the performance of our planners in Section 4.3 and demonstrate a clear winner. In the analysis of the planner, it is clear when boustrophedon coverage is the optimal decision and when it is not.

Multi-Robot Extension

Finally, as is typical in many robot applications, the question must be asked, do "many hands make light work?" In Section 4.4 we answer that question and present extensions to our algorithm.

1.4 Blind Coverage

In the previous formulation, state is known at every point of execution and no steps need be taken for localization. Next, we consider the same problem without movement constraints—using a holonomic vehicle—but both reduce our information state, knowledge of robot pose, and generate a fully offline plan. Most of the existing coverage techniques rely on precise control of the robot's motion. For example, techniques based on the boustrophedon decomposition [30, 90] require the robot to be able to travel accurately in straight lines along the coverage passes, and also to be able to transit precisely between the passes. The motivating example is vacuuming a bounded space. To explore the impact of a partial information state on the coverage problem, we say that the robot only has a bump sensor and compass. The compass is a powerful sensor, because it offers a global frame of reference to combat the accumulation of error. In practice a compass does not perform reliably, but it was shown in [72] that a very short range sensor and map can be used to simulate a compass in this formulation.

Specifically, we consider a robot model with only two movement primitives: First, the robot can rotate in place to face a given direction, though this rotation is subject to some unknown bounded disturbance. Second, the robot can move forward from its current position until reaching the environment boundary. The robot cannot measure the distance traveled (it has no odometer nor clock) nor does it have any other sensors to provide feedback about its motion through the world.

In this model the robot's state is not necessarily ever known exactly, but is maintained as a range of possible states and all planning must take into account that lack of specificity. In the previous formulation, all actions were either coverage actions or translation actions. The sequence most often alternates between coverage actions and translation actions with optimality expressed as the path with least time spent translation. In the second model—what we call blind coverage—a lot of time may be spent on localization actions. The primary alternative, realized with great success in the original Roomba [100], is to move with some a degree of randomness. In that case, one expects the probability of complete coverage to increase as the robot continues its movement, though any guarantees are only probabilistic. In contrast, this blind coverage considers a coverage problem in which a robot that is very simple —with no feedback sensing, and with highly error-prone actuation— can nonetheless *guarantee* to cover a certain portion of its environment.

Our interest in such simple robot models derives both from a practical desire to limit the complexity and expense of robots deployed for such tasks, but also from a desire to understand the underlying information requirements of robot coverage. Our algorithm computes a sequence of motions for the robot to attempt to cover as much of the environment as possible before returning to its start state. As stated above, the algorithm must confront the dual challenges of navigation and coverage:

Navigation with this robot model can be challenging because the available sensor data is so limited, the robot may easily lose track of its own position; coverage with this robot model can be challenging because if the robot does not know its own position with relatively high accuracy, it cannot be certain of which parts of the environment are being covered.

Figure 1.4 shows an example of our algorithm's output, in which the differently-colored shaded regions illustrate regions that can be covered by this approach for varying bounds on the amount of rotational error.

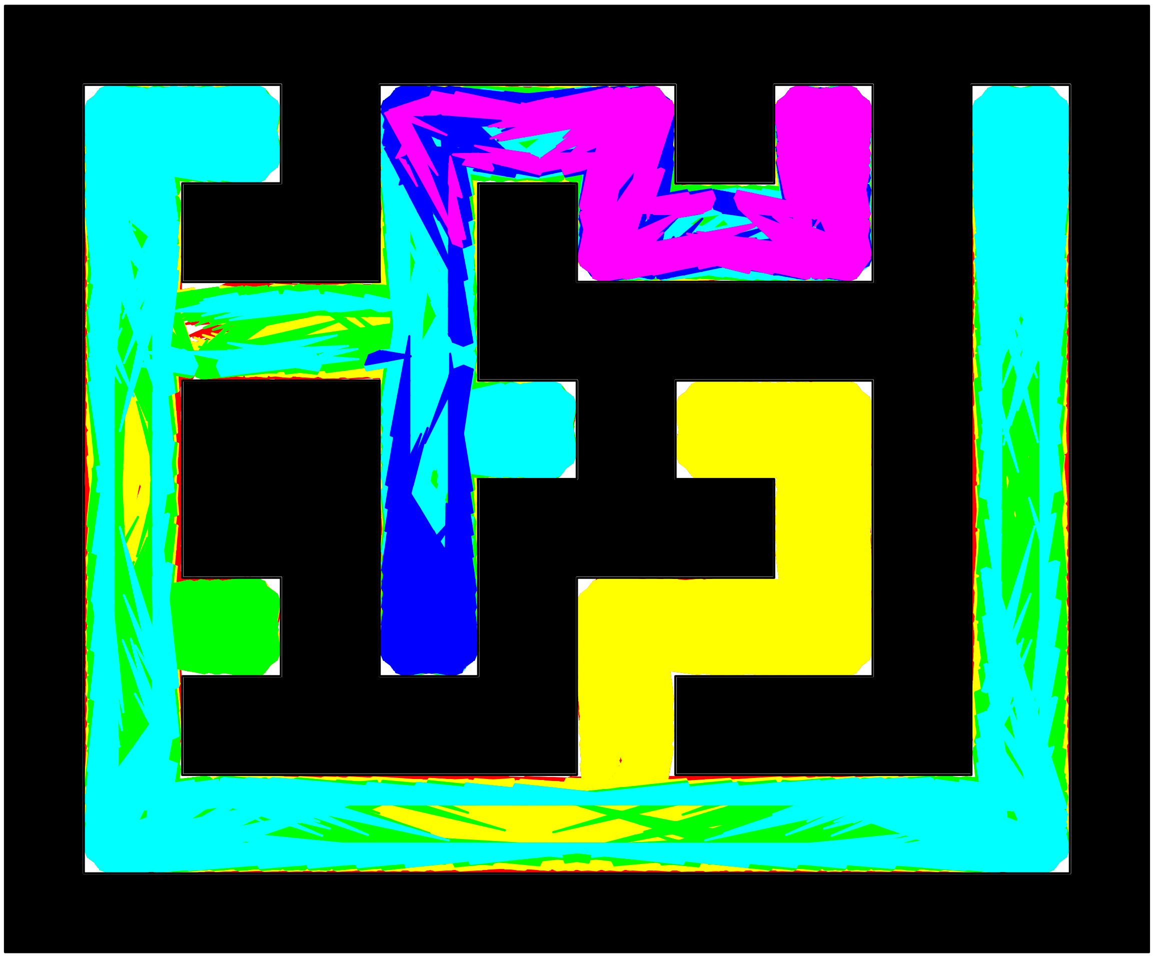

Figure 1.4 A maze-like environment. Our algorithm generates a plan that is guaranteed to cover the portions of this environment. Results from several runs of the algorithm, showing the region guaranteed to be covered by the approach for error bounds ranging from 0.5 degrees to 3 degrees of error on each motion, are shown. For example, the cyan shaded region is the portion of the environment covered by a plan generated by our algorithm for a robot that may experience up to 2 degree of error with each of its motions. The red region (largely occluded by other layers) is the result under a 0.5 degrees per motion error bound.

.

The idea of the algorithm is to construct a directed graph. Vertices of the graph represent contiguous sets of possible states, represented as line segments along the boundary, in which the robot might know its true state lies. Edges of the graph correspond to achievable transitions between these segments, labeled with the region that is guaranteed to be covered by that transition. After constructing this graph, the planning algorithm is then a process of identifying edges that (a) would be beneficial to cross because they would cover some new portion of the environment, (b) can be reached from the starting position, and (c) can be returned from.

Chapter 2

Related Work

2.1 Holonomic Coverage

The various flavors of coverage problems have been studied so extensively that a full review is impossible here. Recent research has studied the role of environment decomposition [3, 46, 55, 108], particularly on grids: [7, 42, 53, 91]; coordination of multiple robots [12,60,61,63,89,90,109]; and different path types such as spirals [24,51] or Dubins curves [61, 71, 95, 106, 107].

We refer the reader to the surveys by Choset [29] and by Galceran and Carreras [47] for a more complete picture. In our initial formulation—Dubins coverage—we discretize and represent the environment in such a way as to make use of our knowledge of the robot's movement constraints to build plans for coverage. Introduced by the work presented in [22, 79], we assume that a map of the environment is provided both for path planning and coverage delineation. We also choose to use some environmental discretization as this is known to be useful in determining when coverage is complete as is pointed out in [29]. It is common to use "seed-spreader" style coverage plans to cover obstacle-free cells [30, 103], therefore we seek a decomposition which lends itself to this pattern yet we do not explicitly make use of it. Though we are looking for a complete coverage algorithm, we do not want online algorithms like [1, 2, 5, 26], since we envision applications of our work repeatedly using the coverage plan.

There are two works which must be addressed in particular. The first [107] by

Xu, Viriyasuthee and Rekleitis inspired our work on an optimal solution to Dubins coverage. The second [106], arrived at the same data structure and search algorithm as our work and therefore need be included for sake of completeness.

2.2 Optimal Holonomic Coverage

Xu, Viriyasuthee, and Rekleitis's planner plans for an environment that is considered to be both obstacle and free-space. We noted that obstacle space in their work is not obstacle in the traditional sense of high cost due to intersection. Therefore, the planner does not explicitly avoid intersection areas marked obstacle. Additionally, each subdivision of the environment is covered nearly independent of others and there are cases where the coverage decomposition must be updated by a potentially non-optimal method. The three major improvements we seek to make to their work is treating obstacle space as space which does not need be covered but offers no additional costs, making plans to cover a decomposed cell dependent on other cells, and avoiding manipulation of cells.

Similar to their work, we begin with a Morse decomposition of the environment resulting in a BCD. Rather than insisting on boustrophedon paths, we allow such paths to emerge from the planner due to them being the "best" thing the robot can do, letting our planner decide exactly how cells should be covered. Our work also differs from [107] in that solutions generated by the planner, contain and are guided by the cost of movement. In [107] a boustrophedon cellular decomposition (BCD) is used, then the edges necessary to visit each cell are produced, but the actual coverage is left to path planning by a boustrophedon path planner. Like Tokekar et al. [98], our algorithm results in a more fine-grain discretization. Rather than sampling however, we slice each cell from the above-described BCD into the number of passes required to cover it following a single axis. This discretization is used to generates plans which allow a robot to cover parts of multiple cells, rather than covering one cell at a time.

The work in [106] also considers the possibility using passes as the nodes of a graph. They do not offer an algorithm to actually generate the graph or a means by which the Hamiltonian cycle will be constructed or implemented. In our work, we provide a planner using the described graph and then implement reductions of that graph to reduce the search space of TSP.

Our work is similar to [55] and [108] in that we are attempting to partition the environment in such a way as to make use of our knowledge of the robot's movement constraints. However, unlike these two works, we are not attempting to minimize the amount of rotation to avoid the kinematic constraint. We, instead, allow the minimum turning radius of the robot to guide our search for good plans. Like [46] the environment is sliced in such a way as to behave as graph-like model, but instead of a spanning tree, a graph is used.

The proposed algorithm can be extended to multi-robot systems by introducing a negotiation protocol such as the one proposed by Kong, New, and Rekleitis [63]. Furthermore, contrary to Acar et al. [3] the generated paths from the proposed algorithm cover all available areas without a need for a backtracking step.

The metric traveling salesman problem with Dubins curve constraints, called Dubins traveling salesman problem, has been well studied [78, 96]. It is defined in the same way as the TSP, but adds a new constraint—that the path to visit all nodes must consist of line segments and curves of a given minimum radius. The Dubins TSP is closely related to our problem. Depending on the formulation of coverage, there is a TSP hiding inside. We show that the path to cover a corresponding coverage problem can be extracted from a plan to visit all the nodes of a Dubins TSP.

The coverage problem for a Dubins vehicle considered in [95] addresses a similar problem and uses a similar vehicle model, but does not use a discretization of the environment and does not provide an algorithmic approach. Our problem differs in that we are very explicit in our decomposition and use it to guide a structured plan to completely cover all area(s) of interest.

2.3 Optimal Coverage With TSP

Because our work builds on theirs, this chapter describes the construction developed by Yu, Roppel, and Hung [106]. The main idea of their work was to solve a coverage problem by mapping coverage to a, so-called, general traveling salesman problem (GTSP). GTSP adds a set of nodesets to the nodes and edges of TSP [77]. A nodeset is simply a set of nodes from the graph. In GTSP, rather than finding the shortest path which visits all the nodes, the goal is to find the shortest path which visits all the nodesets. Noon and Bean provide a construction to turn a GTSP graph instance into an asymmetric TSP (aTSP) graph instance—that is a TSP using a graph in which the edges (v_i, v_j) and (v_j, v_i) may not have the same weight. Commercial solvers and optimizers are available for TSP and the work in [65] provides a mechanism to convert aTSP to TSP, at the cost of doubling the number of nodes and edges of the graph.

Generalized TSP

***Input*:** A weighted graph $G = (V, E)$ and a partition of V into nodesets $S_1, \ldots, S_m$.

***Output*:** The shortest cycle in G that visits each nodeset exactly once.

In their 2015 work, the authors assume a given rectilinear environment with holes, which they call convex [106]. They propose a method by with the environment is discretized into passes which they call "tracks." Their first step is to discretize the environment. Though it is not explicitly described, it appears they envision a sweep line passing along the environment in a direction perpendicular to the direction of coverage to construct their passes. Next, they note that a given track may be covered in either of two directions perpendicular to the sweep line. These two coverage directions become the nodes of a graph and are, each, included in a nodeset for GTSP. They next convert their GTSP graph into an aTSP graph, the result of which can be used to build a plan to cover the environment one track at a time, minimizing the overall cost of time spent not covering the region as show in Figure 2.1.

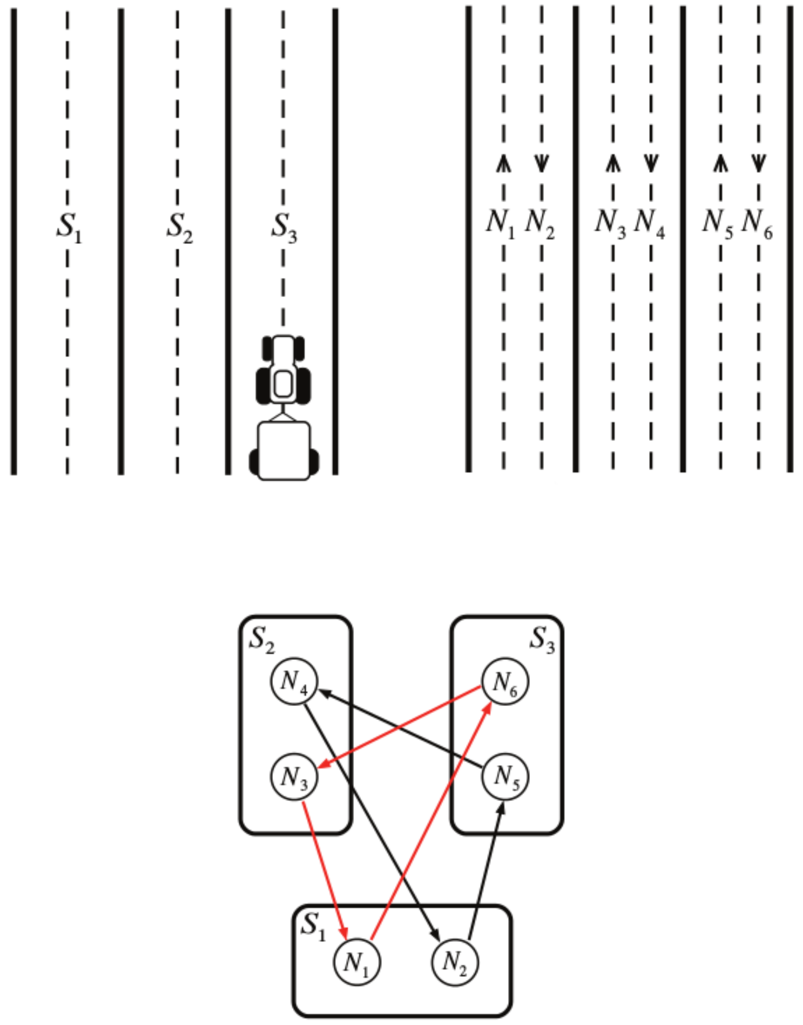

Figure 2.1 An environmental decomposition for a farming robot covering a known bounded region and its translation into an GTSP [106].

The last step the planner must take is conversion from a Hamiltonian cycle to a coverage path plan. Though the authors offer no means by which this should be accomplished, given the nodes of the cycle it is not difficult to construct a plan by interleaving the straight-line passes and Dubins curves between passes into a complete coverage path plan. A planner might start with the node representing the first pass v_0, calculate the straight-line plan to cover it. Next, it would calculate the Dubins curve necessary to translate from v_0 to v_1 and append that to the path and repeat the process for each additional node/edge.

2.4 Minimalist Robotics

Our second formulation is a novel approach to the problem and to the best of the authors' knowledge, there is no work attempting coverage in a similar manner. However, this work draws inspiration from the significant body of prior work on minimalism in robotics.

The idea embodied in this work is related to the idea of "pre-image backchaining" introduced by Lozano-Pérez, Mason and Taylor [74]. Their research describes the notion of a *fine-motion strategy* as an effective counter to position uncertainty in compliant motions. The idea in this research is similar in the way an error cone—a range of possible uncertainty values for each translation made by a robot—will increase the set of possible states from a single known state to some larger set of states derived from a known bound on error.

Erickson, et al. [39] also use the idea of an error cone to solve a global active localization problem. They describe a system whereby actions are carefully chosen to drive the probability of the robot's position toward a single cell in a coarse discretization of the environment. Rather than using a probabilistic approach, a worst-case analysis is used. The other obvious difference is that this is an approach to solving a coverage problem and thus points are used as landmarks, indirectly providing additional information about the robot's state.

The idea of landmark-based navigation was also proposed by Lazanas and Latombe [68]. They suggest the use of landmarks such that while the robot is in proximity of a landmark, the robot is able to execute error-free actions. They also assert that the robot is able to recognize when it has achieved its goal. In contrast, the robot considered here has no sensor which would allow it to do so, nor will planning depend on the robot explicitly sensing that it has achieved its goal state. The planner herein, also, will never assume error-free actions by the robot nor an exact knowledge of any state after leaving the initial state. Instead, a carefully crafted plan that ensures the robot has covered the goal region at plan completion, in spite of its lack of a goal-detecting sensor is used.

There is a similar idea in Erdmann and Mason's sensorless manipulation [36] and Kristek and Shell's deform-able sensorless manipulation [64]. This work will follow suit with an inspection of the robot's environment, rather than any engineering of the environment as in [74]. The synthesis of these works will result in a planner that uses parts of the environment as landmarks, by describing a careful iterative motion process to eliminate uncertainty periodically throughout the robot's execution. By determining landmarks from plentiful environment features, in this case, convex vertices, a very simple robot is able to solve problems previously considered only through changing the environment in some way or the addition of more sensors.

This approach has parallels to prior work on coastal navigation [93], but applies in a *minimalist* setting, considering a robot equipped with no sensors other than a compass and a contact sensor. This study using a very simple robot model is motivated by the obvious desire to understand how navigation problems can be solved with simple, inexpensive robots, but also by a broader interest in understanding what information is truly required to complete the navigation task by minimizing the complexity of the robot.

The goal of considering simplified sensing and actuation systems while solving meaningful problems is not new. A number of different tasks have been addressed with this approach, including manipulation in general [9, 37, 38, 74], part orientation specifically [8, 14, 36, 50, 76, 101], navigation [16, 31, 58, 59, 68, 75], and mapping [1, 26, 27, 79, 99]. More generally, others have explored the question of the minimal sensing requirements to complete a given task [17, 32, 38]. This methodology of minimalist robotics research can arguably be traced back to Whitney [104].

2.5 Blind Navigation

The observations necessary for *blind coverage* come from a previous work by Lewis and O'Kane [72]. In this work, the authors utilize the same robot model—a robot with a single bit bump sensor and compass—to solve navigation problems in a known environment, so-called *blind navigation*. The bump sensor is considered reliable and the compass has a known upper bound on error called $\theta_{\max}$.

As in *blind coverage*, actions are simply to face a given direction and translate until contact with a wall. As described in Section 5.2 and illustrated in Figure 2.2, some actions are labeled safe and some unsafe, where safe is defined as a contiguous set of points along a single environment edge.

The *blind navigation* problem relied a great deal on the authors' so-called "corner-finding" algorithm. A discretization of the environment was necessary, but was built around the idea of bridging the line of sight between two corners which were not "visible" to each other. Uncertainty was allowed to increase by some amount, but then was quickly driven close to zero when a corner was visible from a position which allowed for forward progress in navigation. The discretization, then, was built around structures in the environment which precluded visibility such as T-junction hallways and very long hallways as shown in Figure 2.3 and led to solutions of the form seen in Figure 2.4

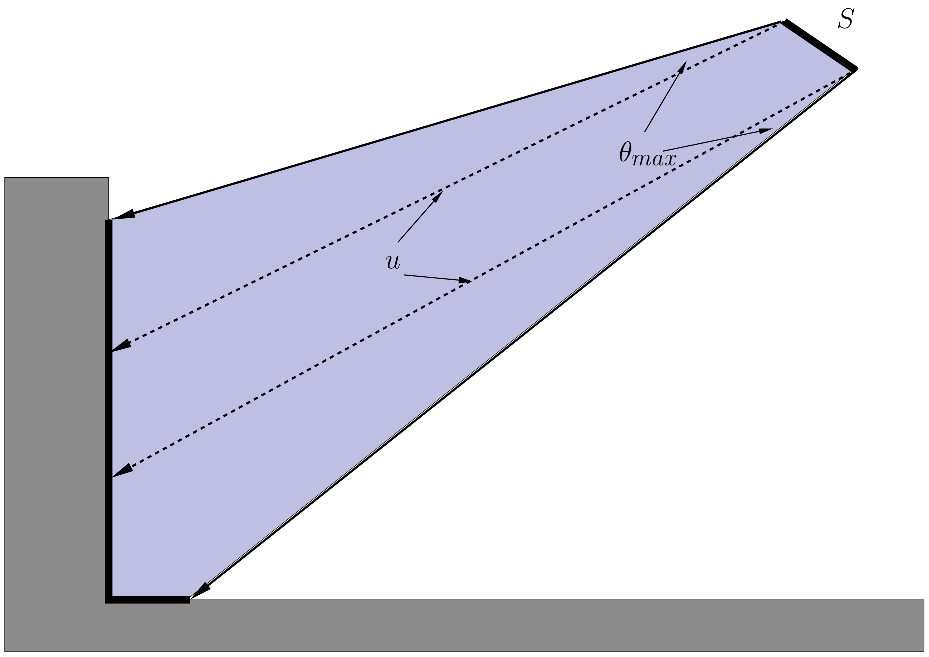

Figure 2.2 An error cone from a state S along action u offset by $\theta_{\max}$ with far boundary spans two edges of the environment. As a result, the illustrated action u would not be in any interval of safe actions. [72]

The biggest differences *blind coverage* and *blind navigation* is that we are solving a different problem—coverage vs navigation. Additionally, we do not make use of their “corner-finding algorithm.” Rather, we allow uncertainty to accumulate and reduce without ever forcing the planner to elect to have the robot localize itself toward a single point at a convex vertex.

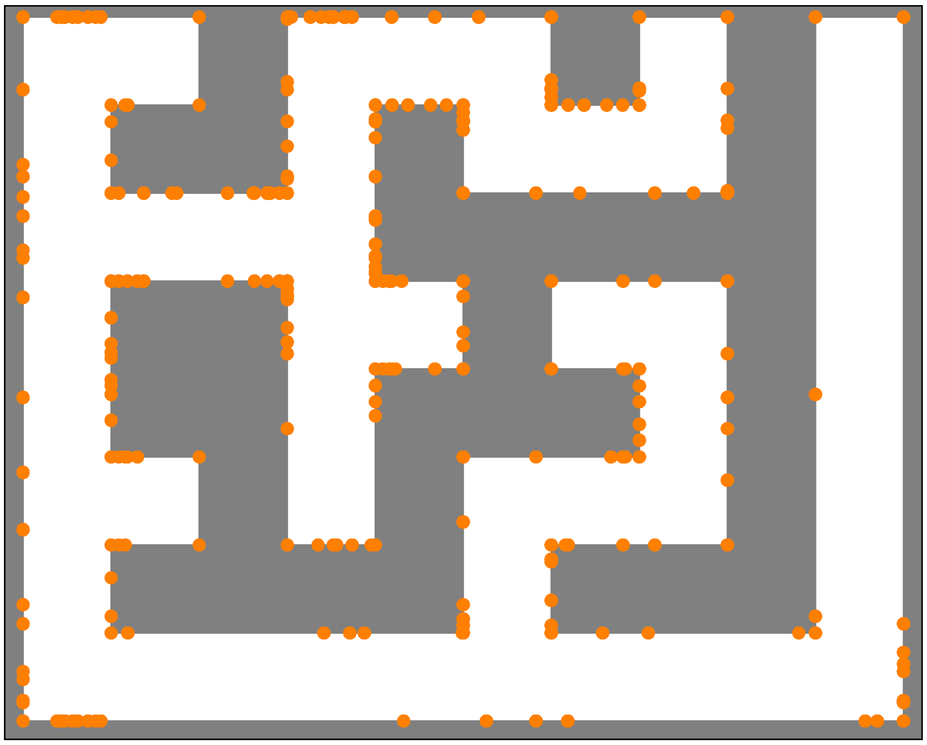

Figure 2.3 An environment environment discretized for blind navigation with corner-finding. [72]

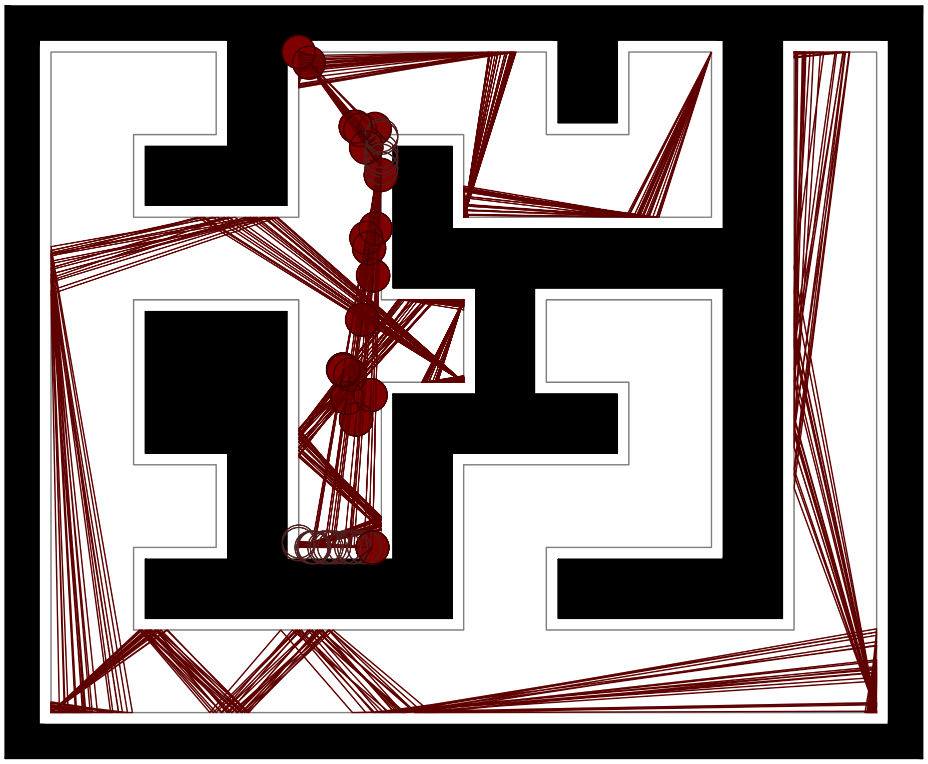

Figure 2.4 A plan generated using the discretized environment from Figure 2.3. The figure shows 20 simultaneous simulations of a robot executing that plan. [72]

Chapter 3

Hardness of Optimal Dubins Coverage

In this chapter, we analyze the difficulty of planning an optimal coverage path for a Dubins vehicle—that is, for a robot that moves forward at a constant speed, constrained by a minimum turn radius. We call this problem *Optimal Dubins Coverage (ODC)*.

3.1 Problem Statement

In this section, we formalize the ODC problem.

Robot Model

This work makes the common assumption that the robot's shape is irrelevant with respect to its workspace. If this is not true, its shape is treated, without loss of generality, as a disk exactly large enough to contain the robot's true geometry. With this assumption, the robot is modeled as a point in a plane with orientation. Its state space is defined as $\mathbb{R}^2 \times [0, 2\pi)$. The robot is described by a discretized time model such that at any time t, its pose is defined as the tuple (x_t, θ_t). The robot's position in the plane is $x_t \in \mathbb{R}^2$ and its orientation is $\theta_t \in [0, 2\pi)$.

The robot's translations are limited by a constraint on its minimum turning radius ρ, and a constraint that movement maintains a constant forward speed, assumed (without loss of generality) to be 1. These constraints result in a vehicle capable of following only Dubins paths [33]. It is important to note that one of a Dubins vehicle's constraints is the vehicle's forward momentum must be maintained. Many path-planning algorithms assume the robot can both stop and back up as a means to get arbitrarily close to an obstacle, before backing away maintaining the minimum turning constraint. Unfortunately we cannot relax the constraint as we would like our planner to handle vehicles like fixed-wing UAVs. Included in the robot's translations, but not explicitly modeled, is some means of maintaining a workable bounds on error. We assume the robot has access to a global positioning system, without requiring that we model the process of bounding error.

The robot has a sensor which allows it to observe a disk centered on its position with radius ϕ. Observations given by this sensor are left purposely vague as they have no bearing on the robot's pose. They are not used explicitly by the planner in any way except to note the area which each would cover. We further make the assumption that sensor readings are continuous or can be taken often enough and that t is small enough that overlapping sensor readings form a region with a static width of ϕ. This assumption is necessary for our planner to achieve coverage.

Figure 3.1 illustrates the robot model notation.

Environment Representation

As is typical in a coverage problem, the planner has a complete and accurate map of the environment or its boundary. From that map, we consider a bounded polygonal subset of the plane, denoted $P \subset \mathbb{R}^2$. Few restrictions are placed on P, as the problem must include possibly disconnected and/or non-convex areas of interest as illustrated in Figure 3.2. Currently there are no areas of obstacle space and translations between disconnected regions are assumed to be possible without consideration of collisions with the environment. This is not an unreasonable expectation as both aerial and aquatic environments often provide such.

Coverage is described as the problem of passing a robot or robot's sensor over

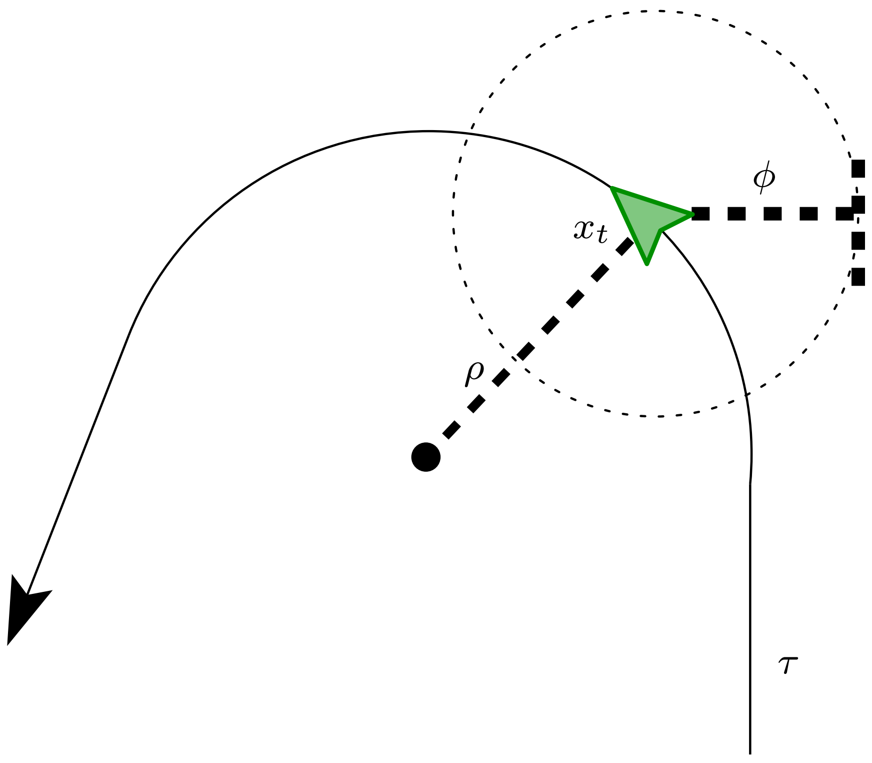

Figure 3.1 The robot executing plan τ, at position x_t, with a minimum turn radius ρ, and a sensor footprint ϕ.

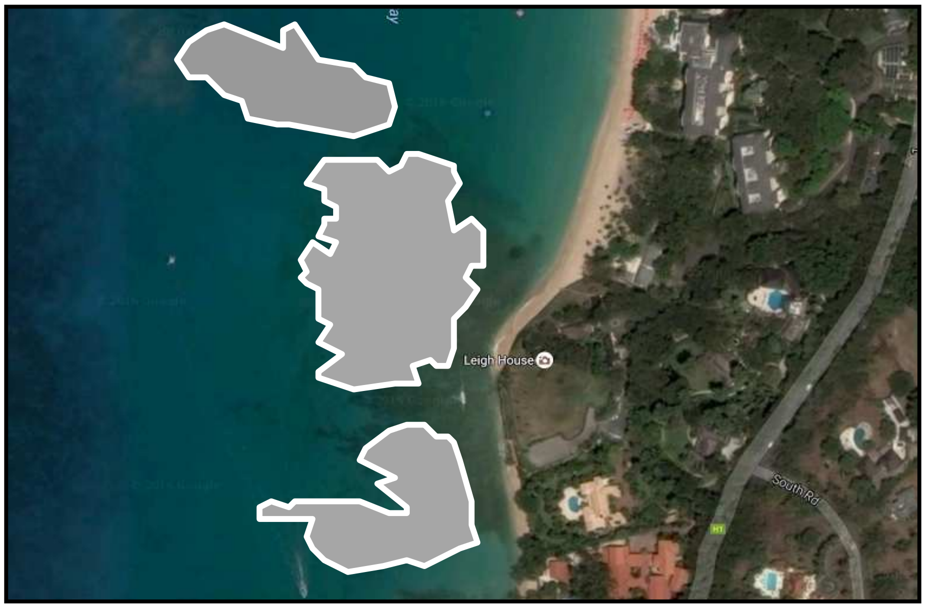

Figure 3.2 A non-convex, disconnected environment solvable by our algorithm. The filled areas outlined in white are areas of interest.

every portion of a known environment. To restate the problem as a motion planning problem, we say that our goal is to generate a plan τ, obeying the robot's motion constraints, defined as

$$\tau\colon\ [0,T] \longrightarrow \mathbb{R}^2 \times [0,2\pi)\,, \tag{3.1}$$

in which T is a finite termination time for the plan. We call τ a *coverage path* if, for every point $p \in P$, there exists a time $t \in [0,T]$ for which

$$|x_t - p| \leq \phi. \tag{3.2}$$

The intuition is that as the robot executes the plan τ, its sensor will pass over every part of the areas of interest. Our goal is to compute an optimal coverage path, in the sense of minimizing the termination time T. We realize the minimization by selecting the shortest path with covers every point in the environment. As there is some fixed minimal coverage necessary for a finite-sized sensor to pass over every point in the environment, the optimization comes in deciding how the robot translates from covering states through non-covering states.

Optimal Dubins Coverage Problem (ODC)

***Input*:** A polygon P, sensor footprint ϕ, and minimum turning radius ρ.

***Output*:** A plan τ of minimum length, which when followed by a robot with sensor footprint ϕ and minimum turning radius ρ results in coverage of the polygon P.

We establish that Optimal Dubins Coverage (ODC), when cast as a decision problem is NP-Complete. Specifically, we consider the following problem.

Optimal Dubins Coverage (Decision Version)

Input: A polygon P, sensor footprint ϕ, minimum turning radius ρ, and $d \in \mathbb{R}^+$.

Output: *YES* if there exists a τ, which when followed by a robot with sensor footprint ϕ and minimum turning radius ρ which results in coverage of the polygon P with length such that $|\tau| \leq d$, *NO* otherwise.

The proof, which proceeds by reduction from Exact Cover [48]—a known NP-complete problem—draws heavily from existing hardness proofs for the Euclidean Traveling Salesman (ETSP) [52, 83] and Dubins Traveling Salesman (DTSP) [78] problems.

In what follows, we write ETSP(Q) to refer to the length of the shortest path that visits each of the points in Q. Likewise, we write DTSP(Q, ρ) for the length of the shortest Dubins curve with turning radius ρ that visits every point in a given finite set of points Q.

The Exact Cover problem from which we reduce is defined thusly.

Exact Cover (EC)

***Input*:** Two families F and F', of subsets of a finite set U.

***Output*:** *YES* if F' is a subfamily of F consisting of disjoint sets, such that F' covers U, *NO* otherwise.

Papadimitriou [83] describes construction that maps an instance of EC to a set points $Q = \{q_1, \ldots, q_m\}$ in the plane with size $O(n^2)$, the problem size of EC. The construction also produces numbers L and δ, and provides two guarantees. If the EC instance has a solution, then

$$\text{ETSP}(Q) \leq L. \tag{3.3}$$

If the EC instance has no solution, then

$$\text{ETSP}(Q) \geq L + \delta. \tag{3.4}$$

Our reduction from EC to DCov utilizes this construction directly. Given an instance of EC, we form an instance of DCov as follows.

1. Execute Papadimitriou's construction to obtain Q, L, and δ.

2. Choose, for the minimum turn radius ρ, any positive value

$$\rho \leq \frac{\delta}{2m\kappa\pi}, \tag{3.5}$$

   in which $\kappa \approx 2.6575$ is the constant the appears in Ny, Feron, and Frazzoli [78].

3. Choose, for the sensor footprint ϕ, any positive value

$$\phi \leq \delta/(2m) - \kappa\pi\rho. \tag{3.6}$$

   Note that Equation 3.5 guarantees the existence of a positive ϕ satisfying this constraint.

4. Select $P = \bigcup_{q \in Q} \mathcal{S}(q, \phi)$, in which $\mathcal{S}(x, r)$ refers to an axis-aligned square centered at x, with diagonal length r.

5. Set

$$d = L + \kappa\lceil n/2 \rceil \pi\rho. \tag{3.7}$$

This construction clearly takes polynomial time. To show that it is indeed a reduction from EC to DCov, we will use three lemmas, two from the literature and one original.

Lemma 1. *(Savla, Frazzoli, and Bullo [96], Theorem 4.2) There exists a constant* $\kappa < 2.658$ *such that, for any finite set of points* Q *and any turning radius* ρ*, we have*

$$ETSP(Q) \leq DTSP(P, \rho) \leq ETSP(P) + \kappa \left\lceil \frac{n}{2} \right\rceil \pi\rho. \tag{3.8}$$

Lemma 2. *(Savla, Frazzoli, and Bullo [96], Theorem 3.1) For two planar poses whose positions are separated by distance* d*, the shortest Dubins curve connecting those poses has length at most* $d + \kappa\pi\rho$.

Lemma 3. *For any set Q of m points in the plane, any sensor footprint ϕ and any minimum turning radius ρ, let $P = \bigcup_{q \in Q} \mathcal{S}(q, \phi)$. Then we have*

$$\mathrm{DCov}\,(P, \rho, \phi) \leq \mathrm{DTSP}(Q, \rho) \tag{3.9}$$

and

$$\mathrm{DTSP}(Q, \rho) \leq \mathrm{DCov}\,(P, \rho, \phi) + 2m(\phi + \kappa\pi\rho). \tag{3.10}$$

Proof. For Equation 3.9, observe that, when the robot is at any point $q \in Q$, its sensor footprint covers all of $\mathcal{S}(q, \phi)$. Therefore, any path which visits each point in Q also covers all of P.

For Equation 3.10, consider a coverage path τ for P. We form a new path τ' identical to τ, except that we insert some additional path segments to guarantee that τ' passes through each $q \in Q$. Because τ is a coverage path for P and $Q \subset P$, we know that for each each $q \in Q$, there exists some t at which the robot's position $x(t)$ passes within distance ϕ of q, that is, $|x(t) - q| \leq \phi$. At this point, we insert into τ' a Dubins curve from $\tau(t)$ to q (with arbitrary orientation) and from this pose back to $\tau(t)$. Lemma 2 ensures that each of these two path segments have length no longer than $\phi + \kappa\pi\rho$. The total length of all of these 'repairs' is therefore bounded by $2m(\phi + \kappa\pi\rho)$, completing the proof. □

We can now state the main result of this section.

Theorem 1. *ODC is NP-hard.*

Proof. Reduction from Exact Cover, using the construction outlined above. We need to show that the EC instance has a solution if and only if the corresponding DCov instance (P, ϕ, ρ, d) has a coverage path of length at most d. We write Q to denote the finite point set generated by Papadimitriou's construction.

Suppose the EC instance has a solution. Then we have

$$\begin{aligned}
\mathrm{DCov}(P,\phi,\rho) &\leq \mathrm{DTSP}(Q,\rho) \\
&\leq \mathrm{ETSP}(Q,\rho) + \kappa\left\lceil \frac{m}{2} \right\rceil \pi\rho \\
&\leq L + \kappa\left\lceil \frac{m}{2} \right\rceil \pi\rho \\
&= d.
\end{aligned}$$

Here we have used, in order, Lemma 3, Lemma 1, Equation 3.3, and Equation 3.7.

For the other direction, suppose the EC instance has no solution. In that case, we know

$$\begin{aligned}
\mathrm{DCov}(P,\phi,\rho) &\geq \mathrm{DTSP}(Q,\rho) - 2m(\phi + \kappa\pi\rho) \\
&\geq \mathrm{ETSP}(Q,\rho) - 2m(\phi + \kappa\pi\rho) \\
&\geq L + \delta - 2m(\phi + \kappa\pi\rho) \\
&\geq L \\
&\geq d
\end{aligned}$$

These inequalities derive from Lemma 3, Lemma 1, Equation 3.4, Equations 3.6 and 3.5, and Equation 3.7 respectively. □

Corollary 1. *DCov is NP-Complete.*

Proof. It remains only to show that DCov is in NP. We can use the coverage path τ as the certificate, and compute the region covered by τ, a finite union of circles and rectangles. Then verify (1) that this region is a superset of P using a standard clipping algorithm, (2) that the length of τ is at most d, and (3) that τ is indeed a Dubins curve for turning radius ρ. □

Chapter 4

Optimal Dubins Coverage

Chapter 3 introduced *Optimal Dubins Coverage* (ODC) and proved that is an NP-Hard problem. We also define, with rigor, the problem we seek to solve. In this chapter, we present our formulation of the ODC problem, a means by which to map an environment to a graph for GTSP, and a family of algorithms to reduce the complexity of the graph to improve the runtime while maintaining a good approximation of the correct answer. Finally we present an extension to a multi-robot formulation in the work by Karapetyan et al. [61]

4.1 Graph Construction

In Chapter 3 we showed that unless $P = NP$, DCov cannot be solved optimally by any polynomial time algorithm. Indeed, the best known algorithm, discussed in Section 2.3, scales quite poorly as the number of passes increases. In addition, the algorithm as originally presented omits a number of geometric details that are essential for a complete implementation. In this section, we introduce an improvement to that algorithm that generates high-quality coverage paths much more efficiently. Algorithm 1 summarizes our approach.

Our first step in constructing a graph from which to generate plans for coverage by a Dubins vehicle is the decomposition of P into pieces for which it is simpler to construct coverage paths a Dubins vehicle can follow. To accomplish this, the planner partitions P into monotone regions using the boustrophedon cell decomposition

Algorithm 1 OPTIMALDUBINSCOVERAGE(P, ϕ, ρ)

```
$\mathcal{D} \leftarrow$ BOUSTROPHEDONCELLDECOMPOSITION($P$)
$\mathcal{P} \leftarrow$ GENERATEPASSES($\mathcal{D}, \phi$)
$V \leftarrow \mathcal{P} \times \{\uparrow, \downarrow\}$
$E \leftarrow V \times V$
$w \leftarrow$ COMPUTEWEIGHTS($E, \rho$)
$(v_1, \ldots, v_n) \leftarrow$ SOLVEGTSP($V', E', w$)
$\tau \leftarrow$ CONSTRUCTPLAN($v_1, \ldots, v_n$)
return $\tau$
```

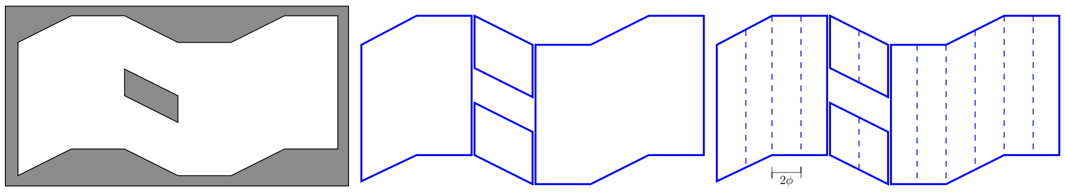

Figure 4.1 Decomposing the environment into passes. [left] The original environment P. [middle] A decomposition of P into 4 cells, each y-monotone, via Boustrophedon Cell Decomposition. [right] A refinement the above decomposition into 14 passes of width at most 2ϕ.

(BCD) algorithm [30] (Alg. 1, line 1). While we do not seek to encode boustrophedon-like coverage behavior, if the planner discovers that is a valid solution, we provide a mechanism for its selection. Next, we further refine the decomposition, cell-by-cell, into a set $\mathcal{P}$ of "passes" $P_i \in \mathcal{P}$ such that each pass is a connected region that can be covered in a single sweep from end-to-end (Alg. 1, line 2). Without loss of generality, we construct these passes with vertical orientations, utilizing divisions parallel to the y-axis; see Figure 4.1. In a practical deployment the choice for the direction of coverage would be affected by a variety of factors, such as, wind- or sea-current direction, desired sensor placement, location of obstacles, etc [107], but would remain axis-aligned.

The result is a set of passes, each no more than 2ϕ wide, such that the robot can cover a pass in a single sweep of its sensor. To cover a given pass P_i, a robot must follow a segment of the pass's vertical bisector. We call this segment the *covering path segment* for the pass. The covering path segment, specified by its top point $t(P_i)$ and

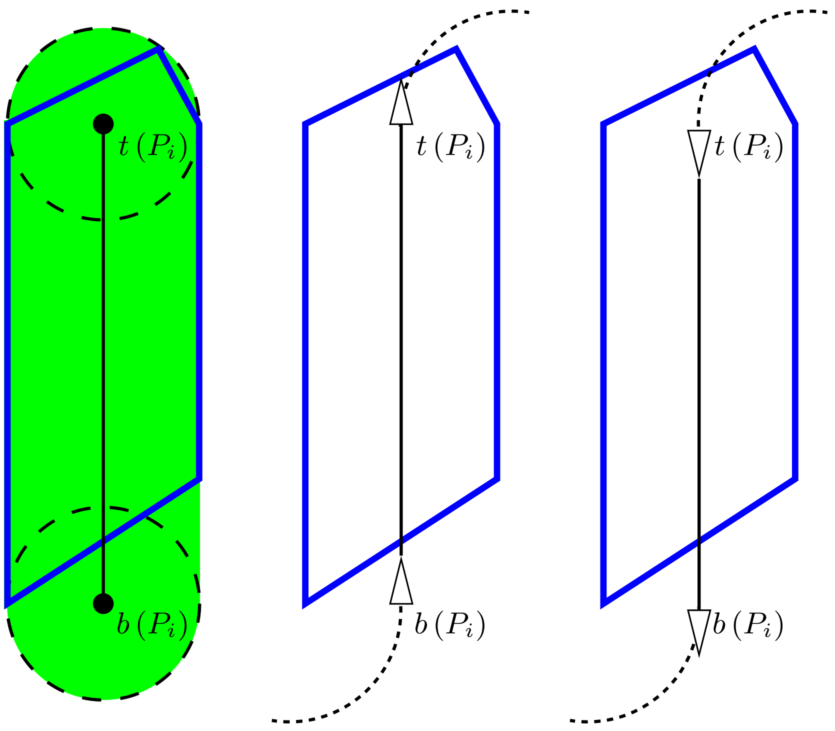

Figure 4.2 [left] A pass P_i and its covering path segment. The shaded region area is covered by the robot's sensor as it traverses the covering path segment. [middle] The graph vertex $(P, \uparrow)$ has entry pose $(b(P_i), \frac{\pi}{2})$ and exit pose $(t(P_i), \frac{\pi}{2})$. [right] The graph vertex $(P, \downarrow)$ has entry pose $(t(P_i), \frac{3\pi}{2})$ and exit pose $(b(P_i), \frac{3\pi}{2})$.

its bottom point $b(P_i)$, is defined as the smallest segment along the vertical bisector of P_i for which

$$P_i \subseteq \bigcup_{q \in \overline{t(P_i)b(P_i)}} \mathcal{B}(q, 2\phi)$$

as illustrated in Figure 4.2. When every pass $P_i \in \mathcal{P}$ is covered, then P is covered.

We must next map our passes into the vertices of a graph and create the necessary edges. Once we have a graph and the solution to a TSP on the graph, we need only map the circuit to a path. These steps are described below.

- The vertex set V consists of $2|\mathcal{P}|$ vertices defined as $V = \mathcal{P} \times \{\uparrow, \downarrow\}$. The interpretation is that visiting a vertex indicates that the robot should cover the given pass by traversing that pass's covering path segment in the given direction. For each *up* vertex $(P_i, \uparrow)$ we define the *entry pose* as $\left(b(P_i), \frac{\pi}{2}\right)$ and the *exit pose* as $\left(t\left(P_i\right), \frac{\pi}{2}\right)$. For *down* vertices $(P_i, \downarrow)$, we define entry and exit poses similarly *mutatis mutandis*. Figure 4.2 illustrates the construction.

- The edge set E contains $4|\mathcal{P}|^2 - 4|\mathcal{P}|$ elements, connecting all pairs of vertices excepting edges to self and edges between pairs in a set. For a given edge $e_{ij} \in E$, its weight w_{ij} is defined as the length of a Dubins curve from the exit pose of the source vertex v_i, to the entry pose of the target vertex v_j (Alg 1, line 5).

- Given a circuit that visits, for each pass P_i, either $(P_i, \uparrow)$ or $(P_i, \downarrow)$, we can directly construct a coverage path τ by alternating *covering path segments* with Dubins curves between the successive passes in the circuit.

To generate the required circuit described above, we solve an instance of the *generalized traveling salesman problem (GTSP)* [77].

Generalized TSP

***Input*:** A weighted graph $G = (V, E)$ and a partition of V into node-sets $S_1, \ldots, S_m$.

***Output*:** The shortest cycle in G that visits each nodeset S_i, exactly once.

This problem is readily shown to be NP-hard by reduction from the standard traveling salesman problem and GTSP instances can be converted to instances of asymmetric TSP (aTSP) by a straightforward linear time construction [66].

As described in Chapter 2.3, we form a GTSP instance, by partitioning the nodes of our graph into $|\mathcal{P}|$ nodesets

$$\{(P_1, \uparrow), (P_1, \downarrow)\}, \ldots, \{(P_{|\mathcal{P}|}, \uparrow), (P_{|\mathcal{P}|}, \downarrow)\},$$

each containing the two complementary vertices for a single pass. This forces a GTSP solution to visit each pass exactly once, either its $\uparrow$ or its $\downarrow$ vertex. While the authors found no solvers for the GTSP, there exist heuristic optimizers for the aTSP are available [73] that are fast in practice (Alg 1, line 6).

The result of SolveGTSP is a circuit providing the order in which each nodeset should be visited. Due to our construction of aTSP from GTSP, the circuit contains each pass in the environment twice, once for the up pass and once for the down pass in either order. To build τ, note that the first occurrence of a pass is the correct direction in which the pass should be covered. For each of these, we link the coverage path segment with a Dubins curve to the first pass in the next node set given in the circuit.

4.2 Graph Reductions

In Chapter 3 we showed that unless $P = NP$, ODC cannot be solved by any polynomial time algorithm. In this section we provide details for a family of practical—but not necessarily optimal—algorithms that more efficiently generate high-quality coverage paths. In all of our approaches, we extend our Algorithm 1 after construction of the graph and calculation of the edge weights on Line 5 we perform a graph reduction before solving the GTSP problem. The process is outlined in Algorithm 2.

The reason we seek to reduce the graph's size is straightforward; even with fast heuristic solvers, GTSP is a hard problem. Performance is directly related to the complexity of the graph, in that the larger the number of vertices and edges, the larger the search space. Intuitively, the more nodes which must be visited the more complex is the circuit to visit them all optimally. Following the same intuition, additional edges create additional options for the path to follow. Additional edges are a boon if they result in a shorter path but offer nothing and extend the search area by a non-trivial amount when they do not.

Using the heuristic graph reductions, we consider several variations to generate subgraphs of G, omitting some vertices and/or edges. These omissions allow us to potentially solve the GTSP more efficiently at the expense of a possible loss of optimality. In each case, we seek to use our domain knowledge to remove edges (or nodes) which would not appear in an optimal circuit of the sets.

1. **Specified directions** — Impose an ordering on the passes in $\mathcal{P}$, with the constraint that all passes generated from a single BCD cell appear contiguously, in order from left to right. For all odd-numbered passes P_i in this ordering, remove the vertex $(P_i, \uparrow)$ and all of its incident edges. Likewise, delete $(P_j, \downarrow)$ for each even-numbered P_j.

   (Our implementation generates the ordering starting from an arbitrary ordering of the BCD cells induced by the underlying geometric data structures. It then sequences the passes within each BCD cell from left to right.)

   The goal of this reduction is to encode the Boustrophedon coverage algorithm into the graph. With this reduction, we expect to see plans which "discover" that strategy.

2. **Alternating directions** — Delete all edges that connect vertices with the same direction. That is, we remove any edge from an $\uparrow$ vertex to an $\uparrow$ vertex, or from a $\downarrow$ vertex to a $\downarrow$ vertex. The effect is to force the robot to alternate between upward and downward sweeps, without pre-specifying the coverage direction for each pass. Our goal is to avoid the specific case illustrated in Figure 4.3

3. **Restricted weights** — Our last observation is that edges which are very long represent longer periods of translation in which no planned coverage is accomplished. As a general means of removing those less useful edges, we first compute the mean μ and standard deviation σ of the edge weights. Next, select a parameter $z \in \mathbb{R}$ and delete all edges whose weight exceeds $\mu + z\sigma$.

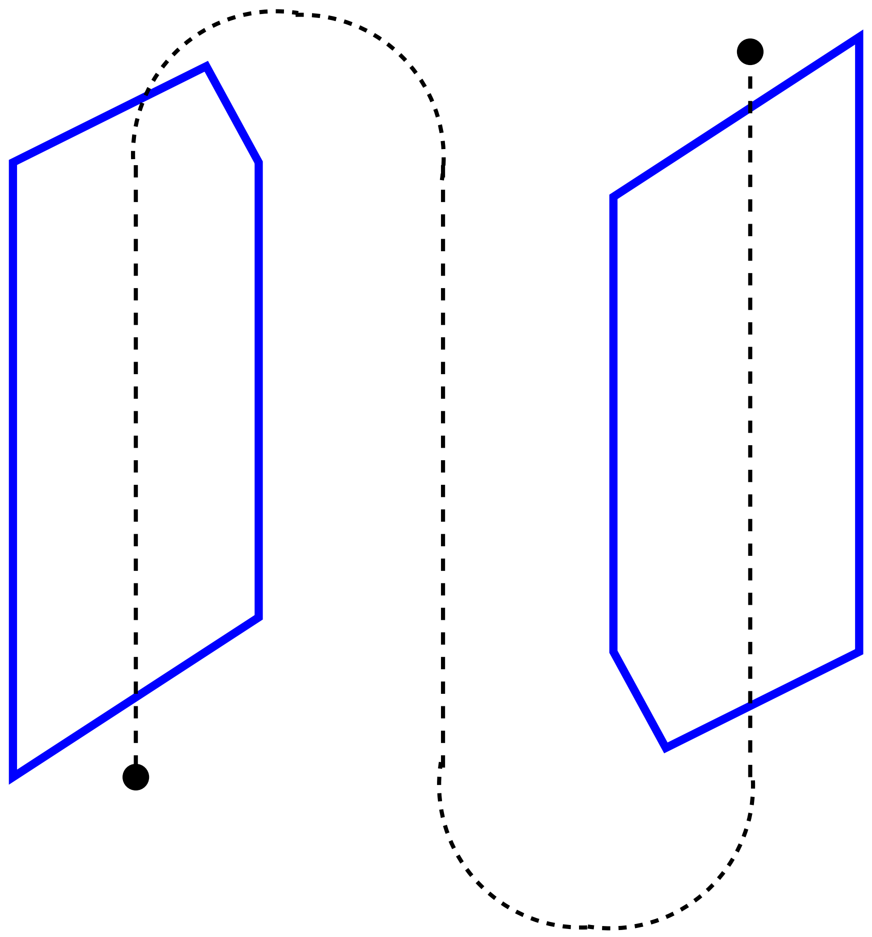

Figure 4.3 A plan to cover to adjacent passes with an unnecessarily long transition path between coverage paths.

4. **Specified directions with restricted weights** — We also consider removing very long edges left after our **specified directions** reduction. The method first removes nodes and edges as described in **specified directions**, followed by removing edges as described in **restricted weights**.

5. **Alternating directions with restricted weights** — In the same manner as above, we seek to reduce the search space of our **alternating directions** reduction by further reduction. In this method we first perform the edge removals as indicated by **alternating directions**, followed by the reduction described by **restricted weights**.

Algorithm 2 EfficientDubinsCoverage(P, ϕ, ρ)

```
$\mathcal{D} \leftarrow$ BoustrophedonCellDecomposition($P$)
$\mathcal{P} \leftarrow$ RefineIntoPasses($\mathcal{D}, \phi$)
$V \leftarrow \mathcal{P} \times \{\uparrow, \downarrow\}$
$E \leftarrow V \times V$
$w \leftarrow$ ComputeWeights($E$)
$(V', E') \leftarrow$ ReduceGraph($V, E, w$)
$(v_1, \ldots, v_n) \leftarrow$ SolveGTSP($V', E', w$)
$\tau \leftarrow$ ConstructPlan($v_1, \ldots, v_n$)
return $\tau$
```

Algorithm 2 summarizes the overall approach. In the next section, we describe simulation results comparing these options.

4.3 Experiments

We implemented the above algorithms in simulation using "a hybrid mathematical programming solver," LocalSolver [73] as our TSP solver. LocalSolver does not solve the TSP exactly and can get stuck in local maxima/minima. As LocalSolver does not compute an optimal answer, it requires a halting value—either time or iterations. Because iterations are linked to the complexity of the problem, we use this as our halting metric. That is, LocalSolver can complete more iterations per second for simpler problems. This gives us a means by which we may judge the reduction in complexity of our different algorithms. In all of the examples below, we chose an arbitrary 5,000,000 iterations as our halting condition.

The algorithms were implemented in C++ and animations were performed with OpenGL. Below we provide results for two of the environments our planner solved. The first environment in Figure 4.4 represents an environment with an area of interest wrapping around an area which is not to be covered. One example is a crop-dusting vehicle covering a similarly shaped field. The second environment in Figure 4.4 contains two holes in coverage. This environment is meant to depict an area in which

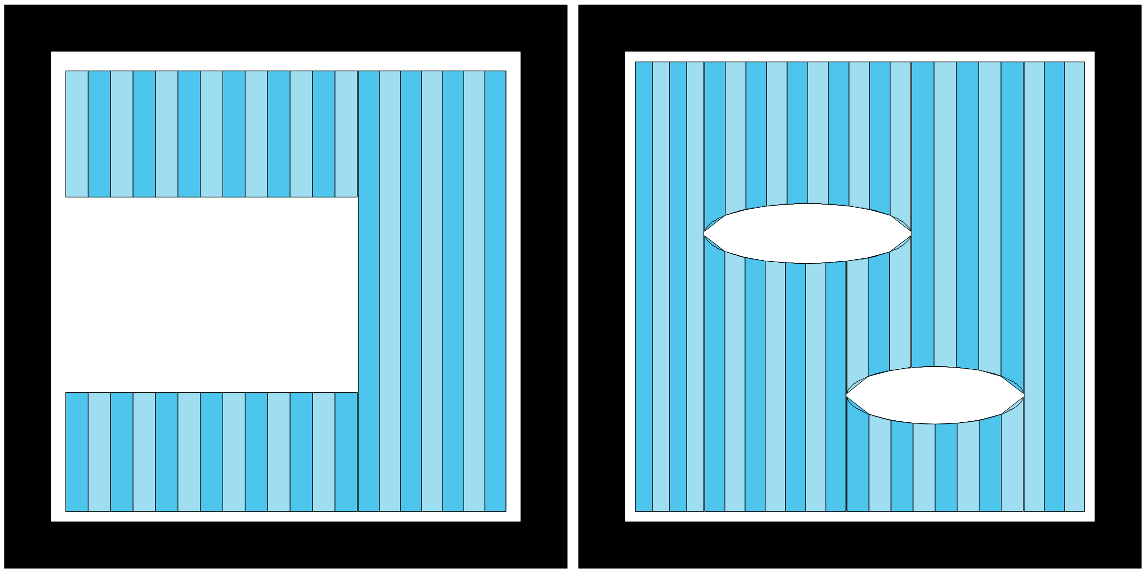

Figure 4.4 [left] An environment wrapping around an area which does not require coverage. [right] An environment with two areas which do not need coverage.

we are mostly interested, but does contain regions with no value for coverage.

We first present the results of Figure 4.5, results of running Algorithm 1 on a full graph and a graph reduced by restricting edges. Minimum turning radius ρ was set to 30.0 and, a relative, sensor footprint ϕ of 5.0 was used. Both solutions τ_1, τ_2 failed at finding a path which does not require a traversal of the length of a pass without covering it. Though the emergent pattern appeared to follow a "way of the ox" [30] coverage, the planner did make use of crossing the holes to shorten its path. With a percent change of -2.59% from the full graph to a restricted edges graph, the planner provides comparable results. The gain from this reduction does not seem worth the time complexity to reduce.

Using the same ϕ and ρ, we ran Algorithm 2 with graphs reduced by both the specified directions and specified directions with restricted weights reductions from Section 4.2; see Figure 4.6 for the resulting paths. Both of these reductions fared better than the two from Figure 4.5. The specified directions reduction had a percent change of -8.68% from a full graph and -1.90% from restricted edge reduction.

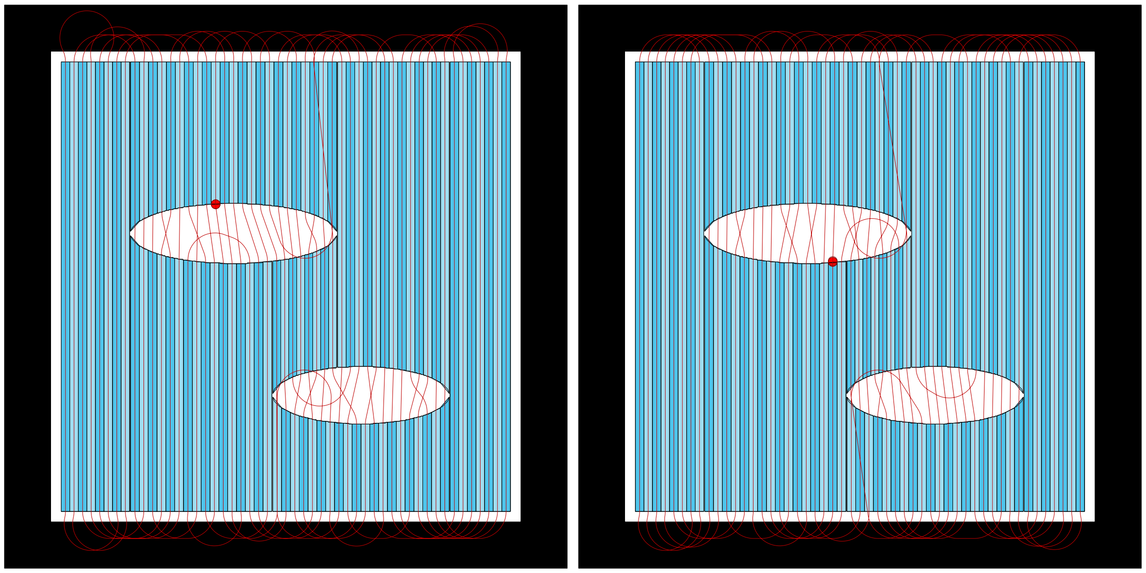

Figure 4.5 [left] A plan of length 37,535 constructed by simulating Algorithm 1 with a complete graph. [right] A plan of length 36,563 constructed by simulating Algorithm 2 with a graph with restricted edges.

Adding the restricted weight reduction did not improve the path length from just using specified directions.

An apparent weakness in our implementation becomes obvious in the alternating directions reduction. We use the reduction from GTSP to aTSP described in [77] and solve the aTSP with LocalSolver. Because LocalSolver is an optimizer and not an exact solver, it gets stuck in local optima. This behavior often leads to paths which violate the Noon and Bean construction from Chapter 2.3. The violation results in paths which do not always visit all node sets of GTSP and therefore do not cover an environment. To provide some implementation of our algorithm running on a graph reduced to alternating edges and alternating edges with restricted edge weights, we had to solve less complex environment and so increased ϕ to 45.0 and kept the ρ from previous experiments. We ran Algorithm 2 with graphs reduced by both the alternating directions with restricted weights and alternating directions with restricted weights reductions from Section 4.2.

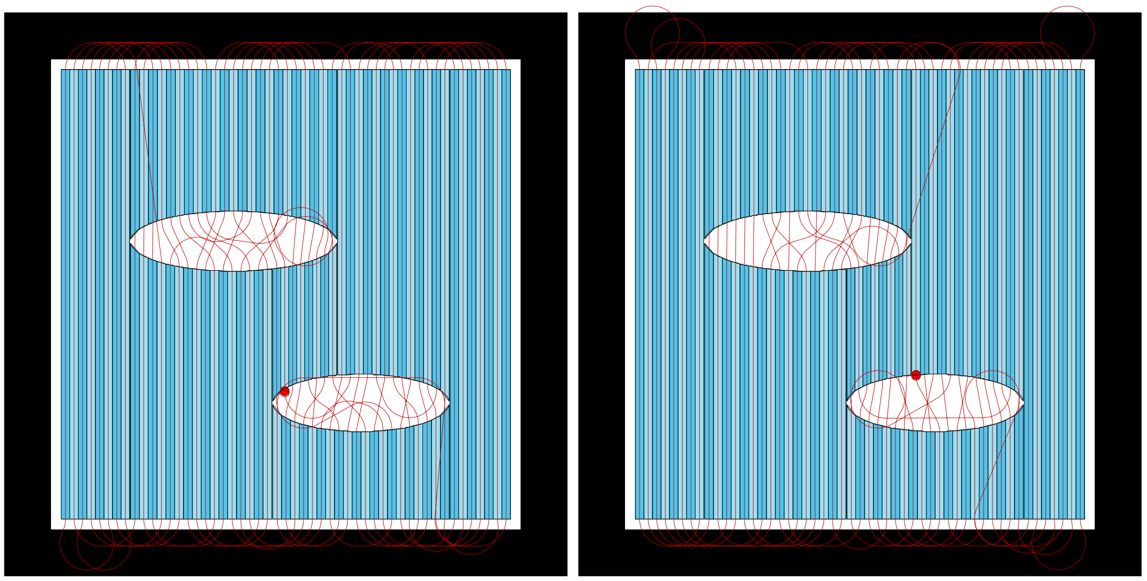

Figure 4.6 [left] A plan with length 34,278 constructed by simulating Algorithm 2 with graph reduction down to specified directions. [right] A plan with length 35,867 constructed by simulating Algorithm 2 with a graph reduction to specified directions with restricted weights.

Figure 4.7 is the result. An interesting emergence from these graph reductions was the boustrophedon path which the robot followed. This path was the original insight the author made from [107]—that given a knowledge of the turning constraints of a robot it might sometimes be better to skip an environment slice and come back to get it later.

Though a visual inspection of paths may provide insight into "attractive" paths versus unattractive ones, data points are necessary to truly grasp the performance of these reduction techniques. In the following set of experiments, we ran all reductions on the two environments shown, varying ρ in Figures 4.8, 4.9 and ϕ in Figures 4.10, 4.11. The experiments in Figure 4.8 and Figure 4.9 held ϕ at a constant 10.0 and varied ρ. The experiments in Figure 4.10 and Figure 4.11 held ρ at a constant 15.0 and varied ϕ. Of the six graph type reductions, only restricted edges, complete, specified directions, and specified directions with restricted edges, were able to regularly solve these two environments under the varying inputs in the computation limit we gave.

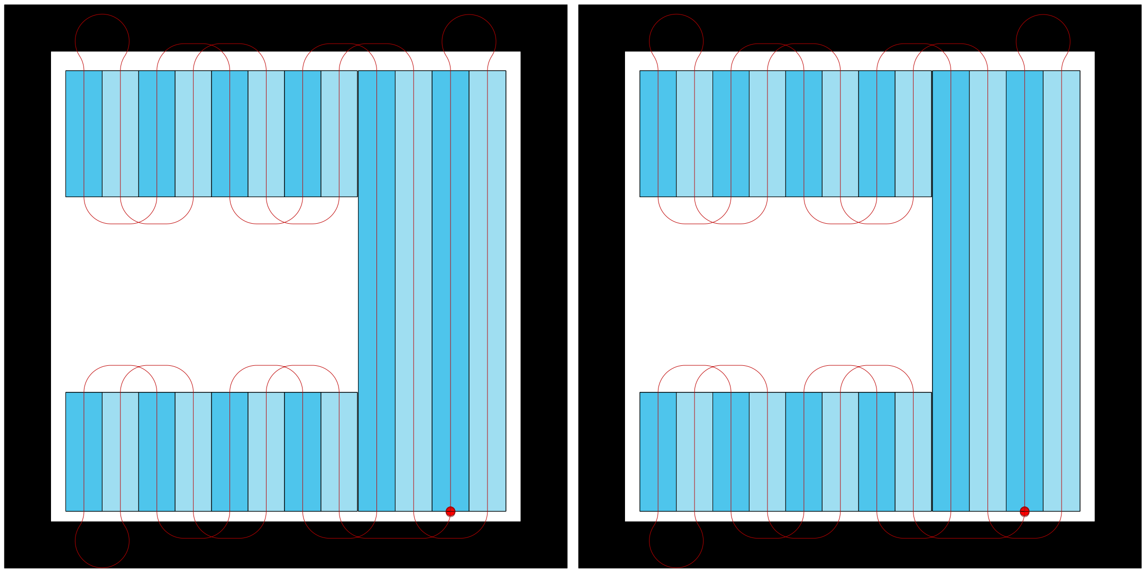

Figure 4.7 [left] A plan of length 3487 constructed by simulating Algorithm 2 on the graph using an alternating directions with restricted weights. [right] A plan of length 3487 constructed by simulating Algorithm 2 on the graph using a reduction to alternating directions with restricted weights.

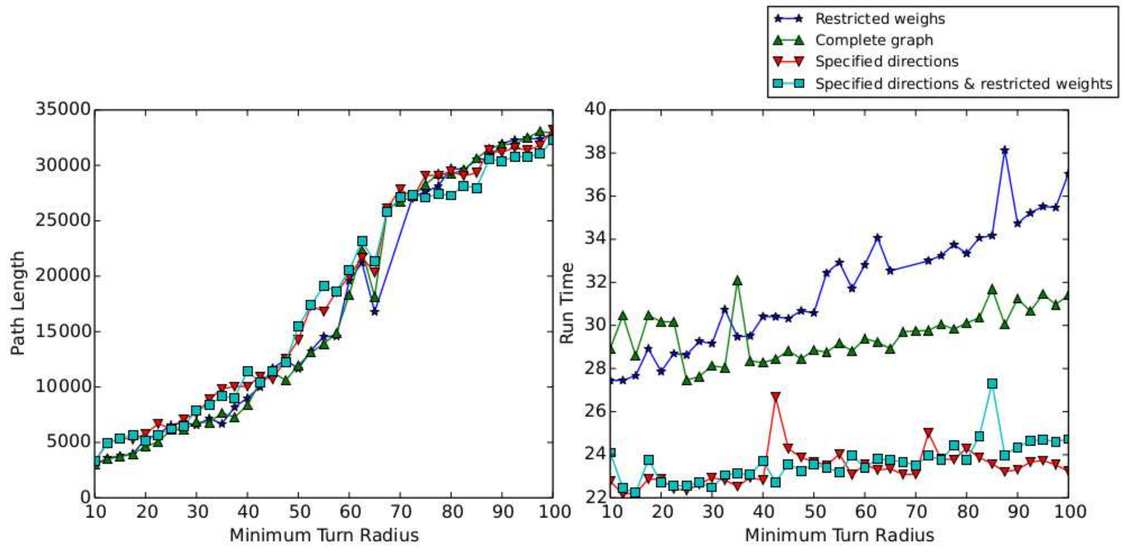

Figure 4.8 The results of covering the left environment Figure 4.4 with ϕ constant at 10.0, varying ρ.

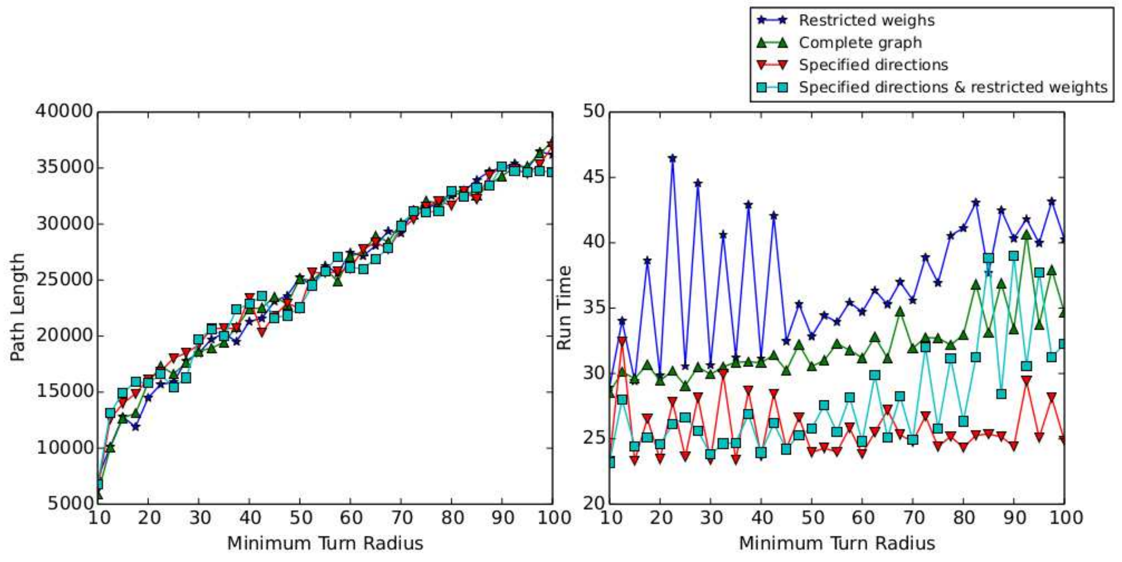

Figure 4.9 An experiment ran to cover the right environment in Figure 4.4 with ϕ constant at 10.0, varying ρ.

In Figures 4.9 and 4.11, results from the reductions to alternating directions and alternating directions with restricted weights appear. The observation is that any chance for LocalSolver to find viable solutions to these reductions occur sparsely and for simpler input values ϕ and ρ.

Finding a clear and obvious winner in specified directions graph reduction, Figures 4.12 and 4.13 we ran experiments to determine how varying both ϕ and ρ affects both path length and runtime.

4.4 Multi-robot Implementation

A solution to the multi-robot coverage problem is important. To cover relatively large areas, meaning areas with small sensor radius relative to the area of coverage, a single robot may not be enough. Figure 4.14 is a satellite image of a large lake near Columbia, SC. The lake has a max length of 66km and max width of 23km. Plans to cover this area could be incredibly long depending on sensor width. With a sensor

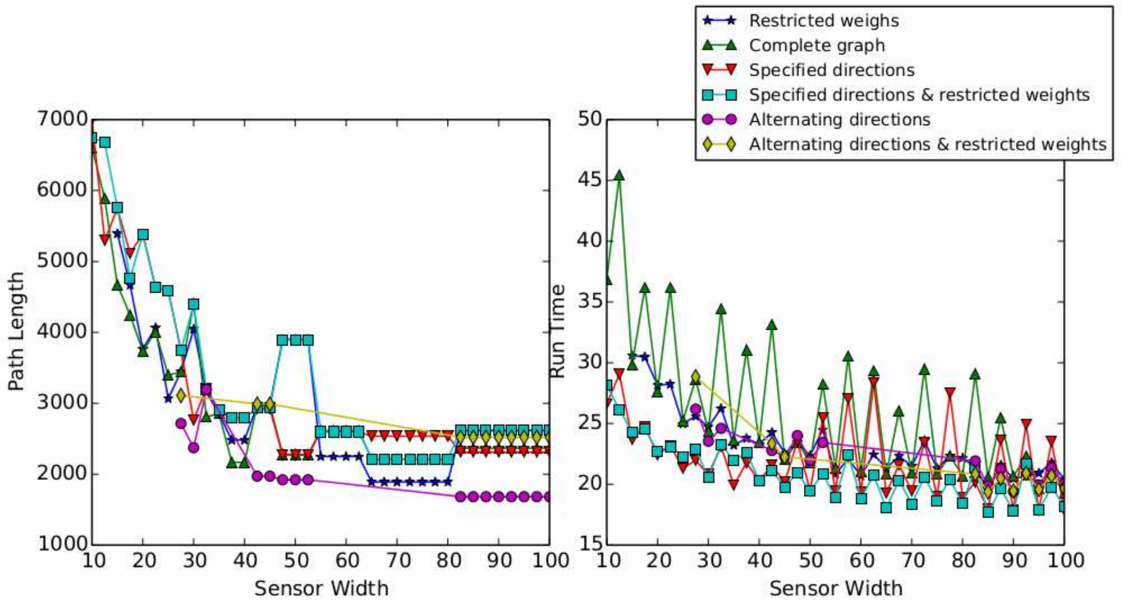

Figure 4.10 An experiment ran to cover the left environment in Figure 4.4 with ϕ varying, and ρ constant at 15.0.

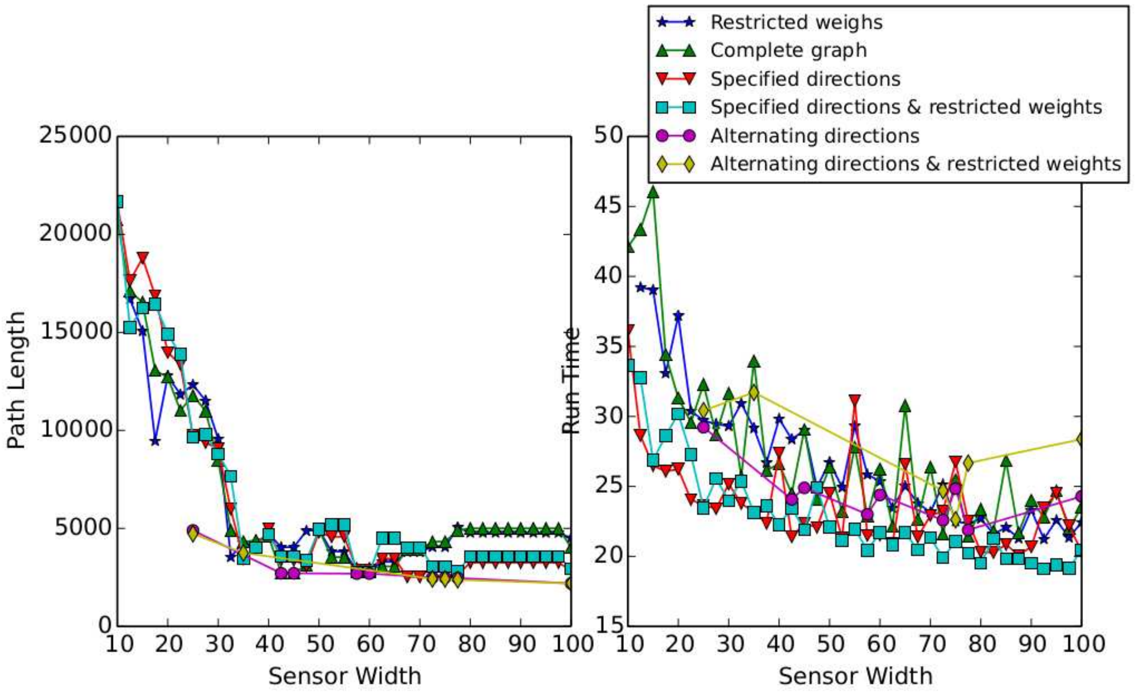

Figure 4.11 An experiment ran to cover the right environment in Figure 4.4 with ϕ varying, and ρ constant at 15.0.

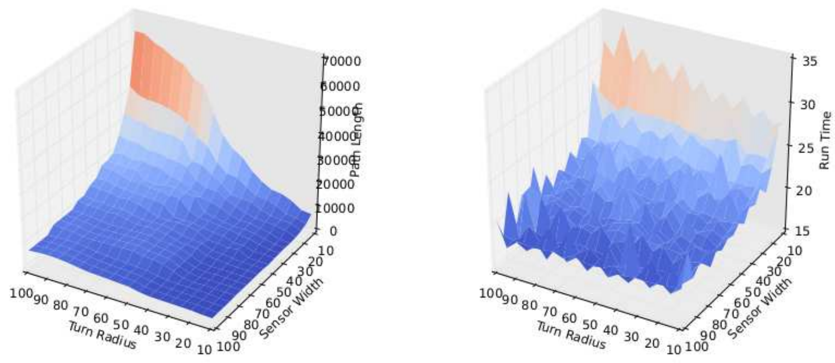

Figure 4.12 An experiment ran to cover the left environment in Figure 4.4 varying both ϕ and ρ.

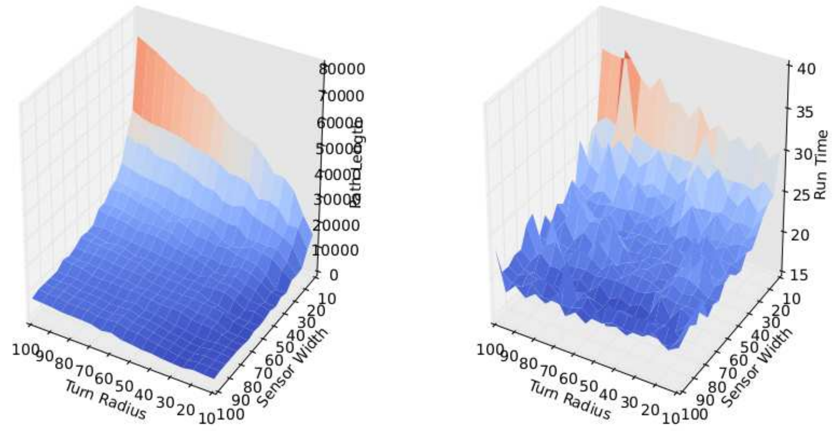

Figure 4.13 An experiment ran to cover the right environment in Figure 4.4 varying both ϕ and ρ.

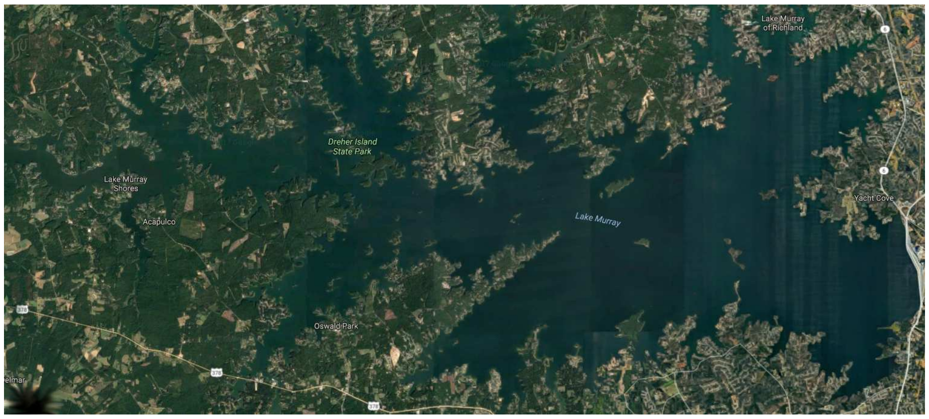

Figure 4.14 Satellite image of Lake Murray near Columbia, SC taken from the Google Map web service. One target for our simulation. A plan to cover the lake with any but the largest coverage radius would result in an extremely long coverage path.

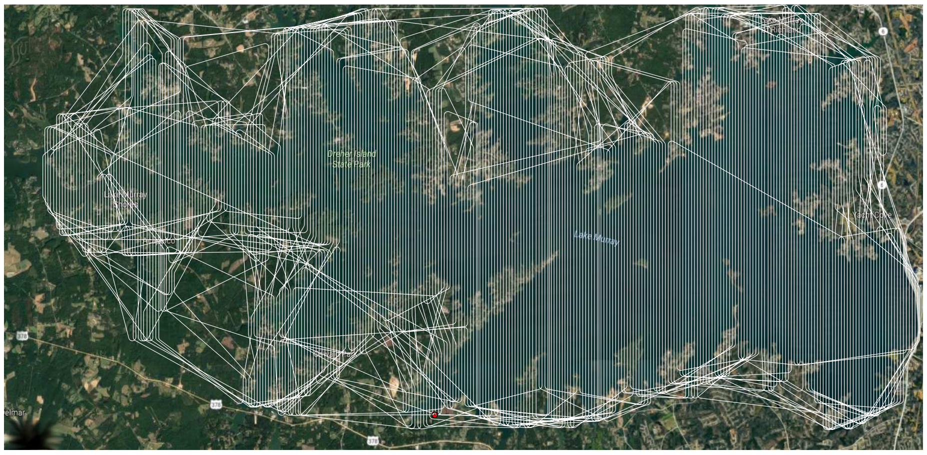

Figure 4.15 A coverage plan for a single robot generated by our Dubins Coverage algorithm.

width of 285m, we generated a coverage plan for the environment and depict it in Figure 4.15. As is evident, even with the large sensor width, the number of passes required to cover the lake are enormous. Without a mobile energy source, it is likely that an actual robot could not, in fact, complete the task.

In a collaboration with Karapetyan et al. [61] and the Autonomous Field Robotics Laboratory (AFRL) [6], we extended the Dubins coverage algorithm to allow multiple robots to cooperate in coverage. In Karapetyan's first work on the multi-robot coverage problem, the author presents her Coverage with Route Clustering (CRC) and Coverage with Area Clustering (CAC) algorithms [60]. In CRC, the environment (as a binary occupancy grid) and number of robots is provided to the algorithm and a tour is generated with algorithm from [107] and that tour is separated into a tour for each robot using a k-postman approximation algorithm by Frederikson, Hecht, and Kim [45]. Using the second algorithm CAC, they first break the environment into a number of "approximately equal partitions" which act as vertices in a graph. Next, they cluster the partitions into one subgraph for each robot and solve the Chinese postman problem (CPP) for each subgraph. The tour of edges is then used to generate a coverage plan, again using [107].

We extended this idea to generate plans for Dubins Coverage that can be executed by multiple robots. The Dubins Coverage algorithm is used extended by both the CRC algorithm as well as the CAC algorithm, resulting in Dubins Coverage with Route Clustering (DCRC) and Dubins Coverage with Area Clustering (DCAC). The first application is a very logical usage of or extension to Dubins Coverage. Dubins coverage is used to produce an optimal tour of vertices to cover the area. Next, the path is subdivided into approximately equally long paths, with consideration of travel time to-and-from the beginning and end coverage tour. Shown in Figure 4.16, the robots' coverage regions overlap in several places, as the planner selected nodes which do not always lead to a Boustrophedon coverage behavior. As expected this extension improves the execution time of any plan by a factor of the number of robots used minus some proportion of the cost to travel to-and-from where their paths start and end.

Where DCAC improves the time to execute plans generated by Dubins coverage, DCAC from Karapetyan et al. [61] not only does the same, but also simplifies the problem making thereby improving the runtime. As in Dubins coverage, the envi-

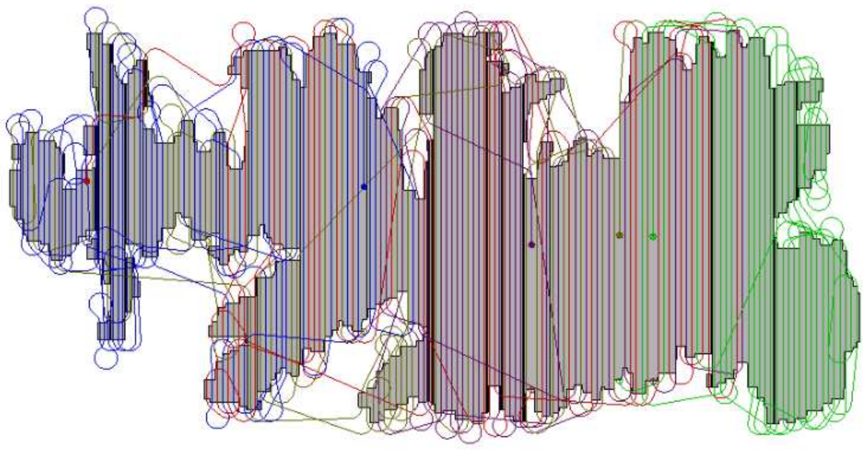

Figure 4.16 A coverage plan using 5 robots to execute a multi-robot coverage plan generated by Dubins Coverage with Route Clustering for the Lake Murray simulated environment.

ronment is first divided into regions monotonic cells, the cells are further divided into passes, the passes are used to generate a weighted, directed graph. Next, however, the graph is divided into k subgraphs, where k is the number of robots. As in CAC [60], each subgraph is built in consideration to how far it lies from the starting point by adding the cost to go and return to the cost to cover. Lastly, like the CAC algorithm DCAC (possibly) uses a heuristic reduction and the solves the GTSP for the graph. Figure 4.17 illustrates the planner solution. It is evident that the DCAC algorithm integrates well with Dubins coverage. The sections of the map allocated to each robot are more tightly grouped than DCRC as expected, and the coverage regions get smaller the further they are from the starting point.

It is interesting to note that this version of the multi-robot extension also has an impact on the run time of the algorithm. Using $O(2^n n)$ space, the Traveling Salesman Algorithm can be solved exactly in $O(2^n n^2)$ time [13, 54] where n is the number of nodes in the graph. Using brute force and no more space, the time complexity is order $O(n!)$ and whether there exists an exact algorithm that runs in $O(c^n)$ for some $c < 2$ is currently an open problem [105]. Though we do not suggest this speedup

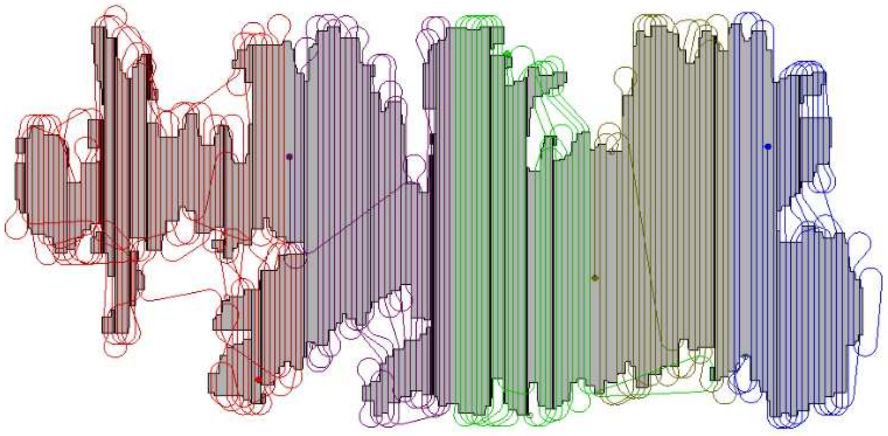

Figure 4.17 A coverage plan using 5 robots to execute a multi-robot coverage plan generated by Dubins Coverage with Area Clustering for the Lake Murray simulated environment.

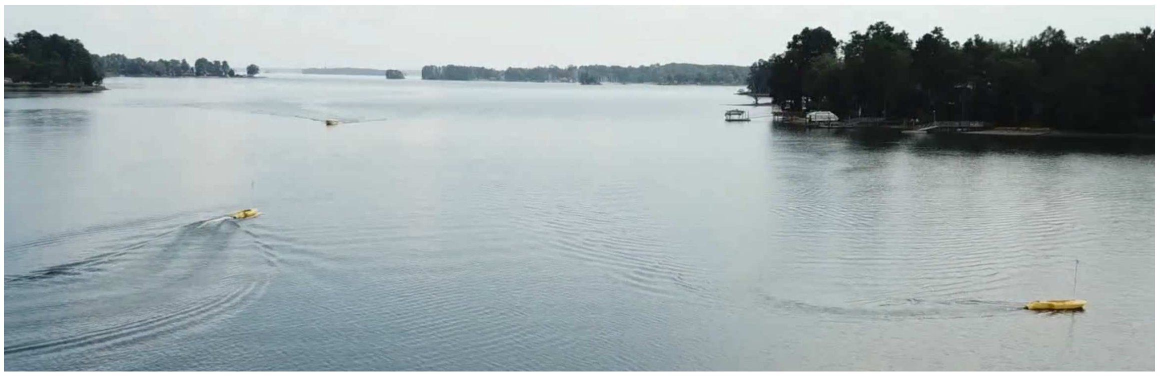

Figure 4.18 Three robots used to execute a plan generated by DCRC.

is a reason to use multiple robots, it is worth noting that a reduction to $k \cdot 2^{\frac{n}{k}} \cdot \frac{n^2}{k^2}$, said another way is $\frac{1}{k} \cdot 2^{n \cdot \frac{k-1}{k}} \cdot (n \cdot \frac{k-1}{k})^2$ nodes for which planning is not required. The paths generated by the DCAC are no longer optimal, but can be generated much faster and have the benefit of a linear decrease in time to execute the plans.

Physical Implementation

Given the superior speed in generating tours from subgraphs using DCAC versus generating a global tour and the subdividing done in DCRC, what is lost is efficiency.

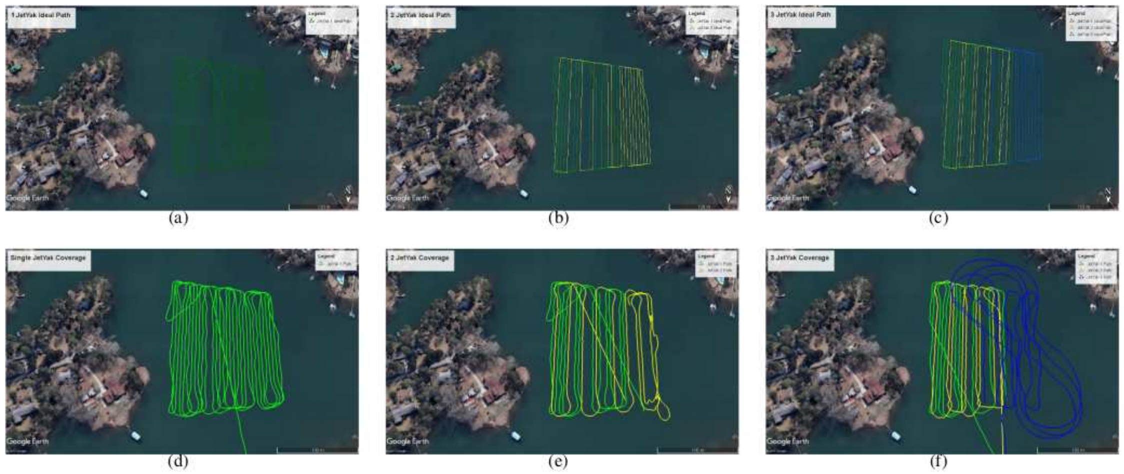

Figure 4.19 The planned paths for 1, 2, and 3 robots are shown in (a), (b), and (c), respectively. The actual paths taken from global information sensors for 1, 2, and 3 robots are shown in (a), (b), and (c), respectively.

DCRC consistently provides a more efficient coverage path than DCAC. Because the cost to cover is so much greater than the cost to create the coverage plan, it was decided for field trials that DCRC would be the planner used. A section of the bottom of Lake Murray near Columbia, SC was covered. A region was selected as map using the Google Maps application. The map was converted into a binary occupancy grid, representing the areas of interest for coverage. The DCRC algorithm was then given the map along with parameters to create tours for 1, 2, and 3 robots. The ideal paths and actual paths are shown in Figure 4.19.

The robots were equipped with a local planner, capable of executing way point following.

Chapter 5

Blind Coverage

As discussed, there is an idea shared by some roboticists—we value solving complex robot tasks with a minimum of robot power. In an absence of powerful sensors and actuators and knowledge, we seek to understand fundamental aspects of the problems put to robots to solve. This chapter details our contribution to a minimalist understanding of the coverage problem.

5.1 Problem Statement

This section provides the details of our robot model and problem.

Robot model

A disk-shaped robot with radius ρ moves through a known, bounded, planar, polygonal environment $W \subseteq \mathbb{R}^2$. Using the center of the robot as its reference point, the configuration space $\mathcal{C}$ is the set of positions within W with distance at least ρ from the boundary of the environment:

$$\mathcal{C} = \{x \in W \mid B(x, \rho) \subset W\}.$$

We follow the usual convention by writing $B(p, r)$ to denote the open ball in $\mathbb{R}^2$ with radius r, centered at p. Note that, though W has a polygonal boundary, the boundary of $\mathcal{C}$ may include both line segments and circular arcs. See Figure 5.1. Informally, the robot's goal is 'drive over' —that is, to move within distance at most ρ of— as much of W as possible.

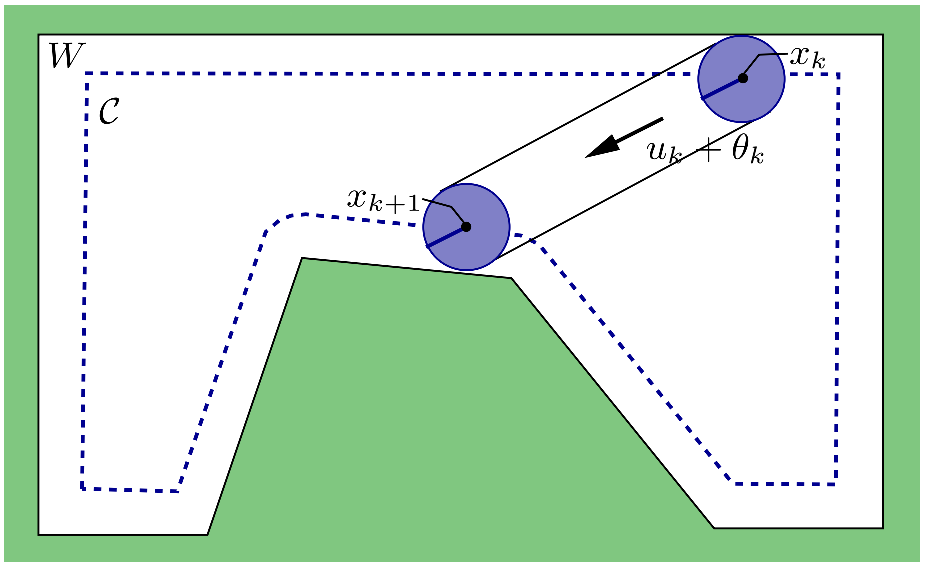

Figure 5.1 An illustration of the basic notation. At stage k, the robot moves in direction $u_k + \theta_k$, from x_k to x_{k+1}, covering a portion of the environment W along the way. Both x_k and x_{k-1} are points along the boundary of $\mathcal{C}$. However, the robot does not necessarily know x_k, and certainly does not know θ_k.

We model time as a series of discrete stages $k = 1, 2, \ldots, K$. The robot's state at stage k is denoted $x_k \in \mathcal{C}$. In each stage, the robot selects a movement direction $u_k \in [0, 2\pi]/\sim$, in which $\sim$ is an equivalence relation that identifies 0 with 2π. This motion is perturbed by an unknown error $\theta_k \in [-\theta_{\max}, \theta_{\max}]$, in which $\theta_{\max}$ is a known bound on the accuracy of the robot's angular orientation. Because we are interested in guarantees of coverage, we do assume that any probability model applies to the selection of each θ_k; the disturbances may be selected at random, or adversarially, or through any other mechanism.

From a given state x_k, the robot moves in direction $u_k + \theta_k$. The motion continues until the edge of the robot's body reaches the boundary of W (or, equivalently, until the center of the robot reaches the boundary of $\mathcal{C}$.) The state resulting from from this motion is denoted x_{k+1}, and we denote this state transition function by f, so that

$$x_{k+1} = f(x_k, u_k, \theta_k).$$

The starting state x_1 is assumed to be known.

This robot model could be implemented, for example, with a robot equipped with a noisy compass and a contact sensor, but no way of measuring the distances it travels. An unusual feature of the model is that, because there is no meaningful feedback from any sensors, the robot's strategy can be fully described as a sequence of motion directions. There is no need to consider any branching or looping in plans executed by this robot.

Minimalist coverage

We can now consider the coverage problem for this type of robot.

Definition 1. *A point $p \in W$ is* covered *by a given sequence of actions $u_1, \dots, u_K$ and disturbances $\theta_1, \dots, \theta_K$ if there exist $k \in \{1, \dots, K\}$ and $\alpha \in [0, 1]$ such that*

$$||p - (\alpha x_k + (1 - \alpha)x_{k+1})|| \leq \rho.$$

Note that Definition 1 refers to a specific sequence of disturbances, and recall that the specific disturbance values are unknown to the robot. Thus, we are interested, as the next definition clarifies, in points that we can guarantee are covered, regardless of the specific disturbances in any particular execution.

Definition 2. *A point $p \in W$ is* certainly covered *by a given sequence of actions $u_1, \dots, u_K$ if p is covered by that action sequence under any disturbance sequence $\theta_1, \dots, \theta_K$.*

Definition 3. *The* certainly covered region*, denoted* $\mathrm{CCR}(u_1, \dots, u_K)$*, is the set of points in W that are certainly covered by $u_1, \dots, u_K$.*

The goal is to select actions that certainly cover some desired fraction of the environment. Specifically, the problem is:

Given an environment W, a start state x_1, a robot radius ρ, and the error bound $\theta_{\max}$, **select** a sequence of actions $u_1, \ldots, u_K$ **to maximize** $\text{Area}(\text{CCR}(u_1, \ldots, u_K)) / \text{Area}(W)$.

5.2 Safe Actions and Possible States

Because of the unknown disturbances, as the robot moves through W, it will in general be uncertain of its position. In our approach, we reason about this uncertainty using a worst-case model. That is, we keep track of which states are possible, based on the history, and which are not.

Specifically, we say that a state $x \in \mathcal{C}$ is a *consistent with* a series of actions $u_1, \ldots, u_k$ if there exists some sequence of disturbances $\theta_1, \ldots, \theta_k$, under which the robot's final position x_k is equal to x. In our approach, we follow our own precedent [72] by considering only plans for which the set of states consistent with the action history is a line segment along the boundary of $\mathcal{C}$. We write $\overline{p_k q_k}$ to denote this segment of possible states at stage k. For consistency, we use the naming convention that a positive rotation of the vector $q_k - p_k$ about p_k is into W. When the robot's position happens to be known with certainty (as happens, for example, before the first action is executed) then $p_k = q_k$ and the segment is a single point.

We say that an action u_k is *safe* from a segment $\overline{p_k q_k}$ along the boundary of $\mathcal{C}$ if the resulting set $\overline{p_{k+1} q_{k+1}}$ of possible states for stage $k+1$ is likewise a segment along the boundary of $\mathcal{C}$. See Figure 5.2.

Given a segment of possible states $\overline{p_k q_k}$ and the next action u_k, we can use the following procedure to simultaneously test whether u_k is safe from $\overline{p_k q_k}$ and, if so, to compute $\overline{p_{k+1} q_{k+1}}$. First, we define a function $\text{ShootRay}(x, u)$ which returns the first point of intersection with δC from a ray emanating from the point x in the direction u. This is a standard operation from computational geometry [23, 97]. To account

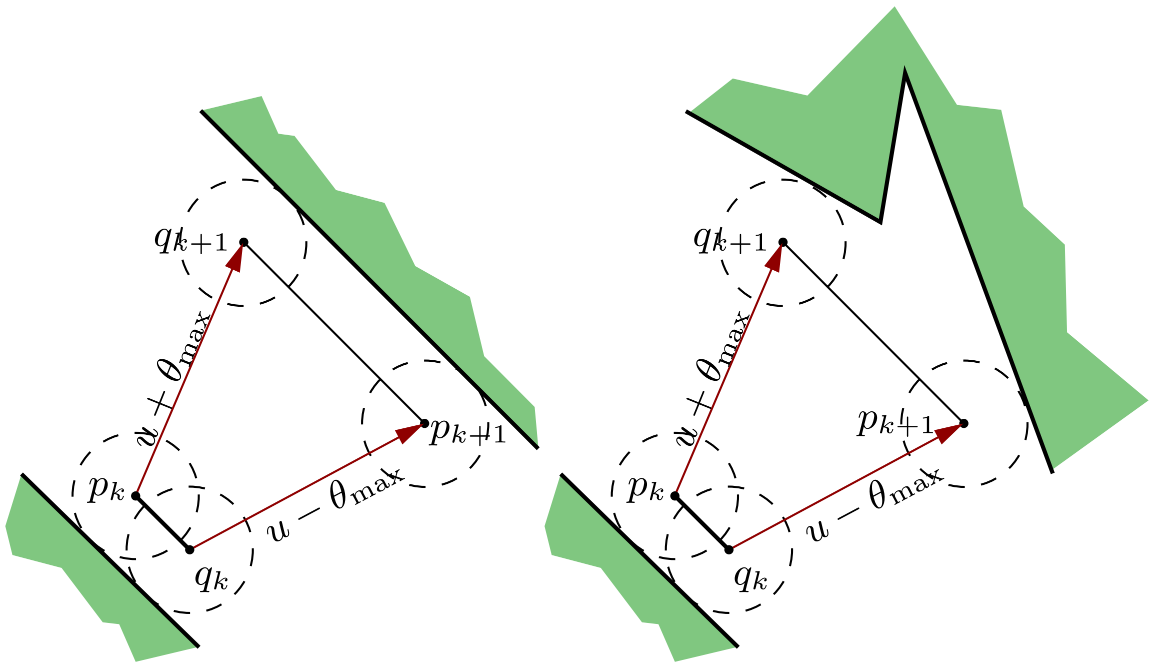

Figure 5.2 [left] An example of a safe action. [right] This action is unsafe, because p_{k+1} and q_{k+1} lie on different edges of the boundary.

for all possible disturbances, $\overline{p_{k+1}q_{k+1}}$ is calculated from $\overline{p_k q_k}$ as follows:

$$\begin{aligned} p_{k+1} &= \text{ShootRay}(q_k, u - \theta_{\max}) \\ q_{k+1} &= \text{ShootRay}(p_k, u + \theta_{\max}) \end{aligned}$$

Next, we test to ensure that the area through which a translating robot may attempt to pass between $\overline{p_k q_k}$ and $\overline{p_{k+1}q_{k+1}}$ is fully within $\mathcal{C}$. A quadrilateral is formed by $p_k q_{k+1} p_{k+1} q_k$ and each edge is checked against $\delta\mathcal{C}$ to ensure no intersections exist. It is also necessary to ensure the quadrilateral contains no vertices of $\mathcal{C}$ to ensure no holes are fully contained within. If the quadrilateral is indeed empty, and if p_{k+1} and q_{k+1} lie on the same segment of the boundary of $\mathcal{C}$, then u_k is safe, and we return $\overline{p_{k+1}q_{k+1}}$. Otherwise, we declare u_k unsafe. (A similar algorithm originally appeared in the context of the navigation problem for a similar robot model [72].)

5.3 Characterizing The Certainly Covered Region

Before considering the broader question of choosing sequences of actions to cover the environment, we must first characterize how the CCR changes as the robot moves. Specifically, in this section, we present two results, one positive and one negative. First, in Section 5.3, we show how to compute the set of states that are certainly covered by a given motion of the robot. Then, in Section 5.3, we state a condition under which a set of points can never be certainly covered by any action sequence.

The region covered by a single movement

Suppose that, at stage k, we know that the robot's state x_k lies within some segment $\overline{p_k q_k}$ along the boundary of $\mathcal{C}$. From there the robot executes action u_k. What can we say about the points, if any, that are certainly covered by this motion? We must be assured that from any point along that segment and at any perturbation of motion, that point would be covered.

By Definition 2 we would appear to need to reason about each of the infinitely many possible disturbances θ_k to establish that a point is certainly covered. Fortunately, we can show that it is sufficient to consider only the extremal disturbances $-\theta_{\max}$ and $+\theta_{\max}$ instead.

Before stating the result, we need the following preliminary definition.

Definition 4. *Given two points p and p' and a radius r the* stadium *between p and p' with radius r, denoted* $\mathrm{Stad}(p, q, r)$ *is the locus of points within distance r of any point along the segment $\overline{pp'}$.*

Visually, the stadium between p and q is a rectangle bisected by the segment $\overline{pq}$, capped by two semicircles of radius r centered at p and q (⬬).

Now we can describe the region covered by a single motion.

Theorem 2. *Suppose the robot has executed a sequence of safe actions* $u_1, \ldots, u_{k-1}$*. Let segment* $\overline{p_k q_k} \subset \mathcal{C}$ *denote the segment of possible states at stage* k*. Consider a safe action* u_k*, and let* $\overline{p_{k+1} q_{k+1}}$ *denote the segment of possible states resulting from this motion. Then*

$$\begin{aligned} CCR(u_1, \ldots, u_k) = CCR(u_1, \ldots, u_{k-1}) \\ \cup \left(\text{Stad}(p_k, p_k', \rho) \cap \text{Stad}(q_k, q_k', \rho) \right). \quad (5.1) \end{aligned}$$

Proof. First, note that for any p, if $p \in CCR(u_1, \ldots, u_{k-1})$, then $p \in CCR(u_1, \ldots, u_k)$. Thus, we need only to consider the points certainly covered by the motion from x_k to x_{k+1}. Let R denote this set. We must show that $R = \text{Stad}(p_k, p_k', \rho) \cap \text{Stad}(q_k, q_k', \rho)$.

($\subseteq$) Let $p \in R$. Note that, since p is certainly covered by this motion, it must be specifically covered in the case where $x_k = p_k$ and $\theta_k = \theta_{\max}$. Thus, $p \in \text{Stad}(p_k, p_k', \rho)$. A similar argument shows that $p \in \text{Stad}(q_k, q_k', \rho)$.

($\supseteq$) Let $p \in \text{Stad}(p_k, p_k', \rho) \cap \text{Stad}(q_k, q_k', \rho)$. We need to show that $p \in R$, which means that for every possible starting point $x_k \in \overline{p_k q_k}$ for the motion, and every possible disturbance $\theta_k \in [-\theta_{\max}, +\theta_{\max}]$, the robot passes within distance ρ of p. The set of locations from which this occurs, for a particular x_k and θ_k, is $\text{Stad}(x_k, f(x_k, u_k, \theta_k), \rho)$. Because this must hold for all x_k and θ_k, we know that if

$$p \in \bigcap_{x_k} \bigcap_{\theta_k} \text{Stad}(x_k, f(x_k, u_k, \theta_k), \rho),$$

then $p \in R$. However, this intersection is fully determined by the two extremal stadia $\text{Stad}(p_k, p_k', \rho)$ and $\text{Stad}(q_k, q_k', \rho)$, which are known by construction to contain p. Thus p is also in R. □

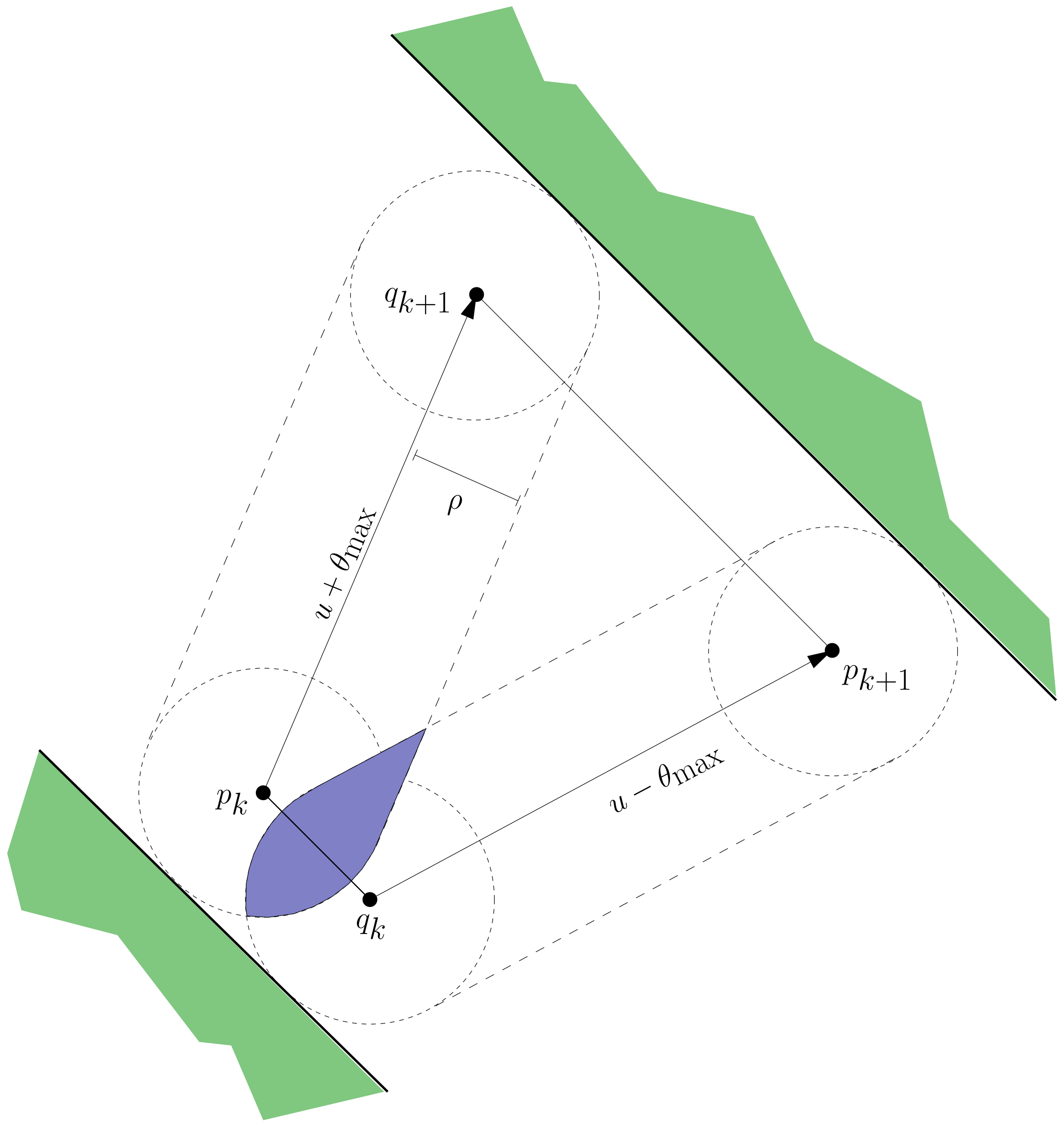

Figure 5.3 Computing the CCR for a single safe action, as described in Theorem 2.

Theorem 2 leads directly to an algorithm for computing the CCR achieved by any motion sequence: Start from the empty set, and iterate over the actions. At each step, compute the union of the previously covered region with the intersection of stadia described in Equation 5.1.

Regions that cannot be covered

We can use a similar idea to the proof of Theorem 2 to rule out certain states from being certainly covered by *any* sequence of motions.

Theorem 3. *Given a point $p \in W$, an error bound $\theta_{\max}$, and a robot radius ρ, let q denote the nearest point on the boundary of W to p. If*

$$||p - q|| > \rho \frac{\tan \theta_{\max} + 1}{\tan \theta_{\max}}, \tag{5.2}$$

then p cannot be certainly covered by any motion sequence.

Proof. Theorem 2 characterizes the region certainly covered at each step as the intersection of two stadia. This intersection is largest when the robot begins at a known position (that is, when $p_k = q_k$) and extends the furthest into the interior of W when the motion direction u_k is perpendicular to the environment boundary. Thus, if p can be certainly covered at all, it can be certainly covered starting at x_k and moving directly toward p. It is a simple matter of trigonometry to determine that the most distant point this region has distance $\rho\frac{\tan \theta_{\max}+1}{\tan \theta_{\max}}$ from q. See Figure 5.4. $\square$

The intuition is that by imagining the robot at the point nearest to p, with no position uncertainty, we construct the best-case opportunity to include p in the CCR. If p cannot be certainly covered under those ideal conditions, then there is no hope to certainly cover p.

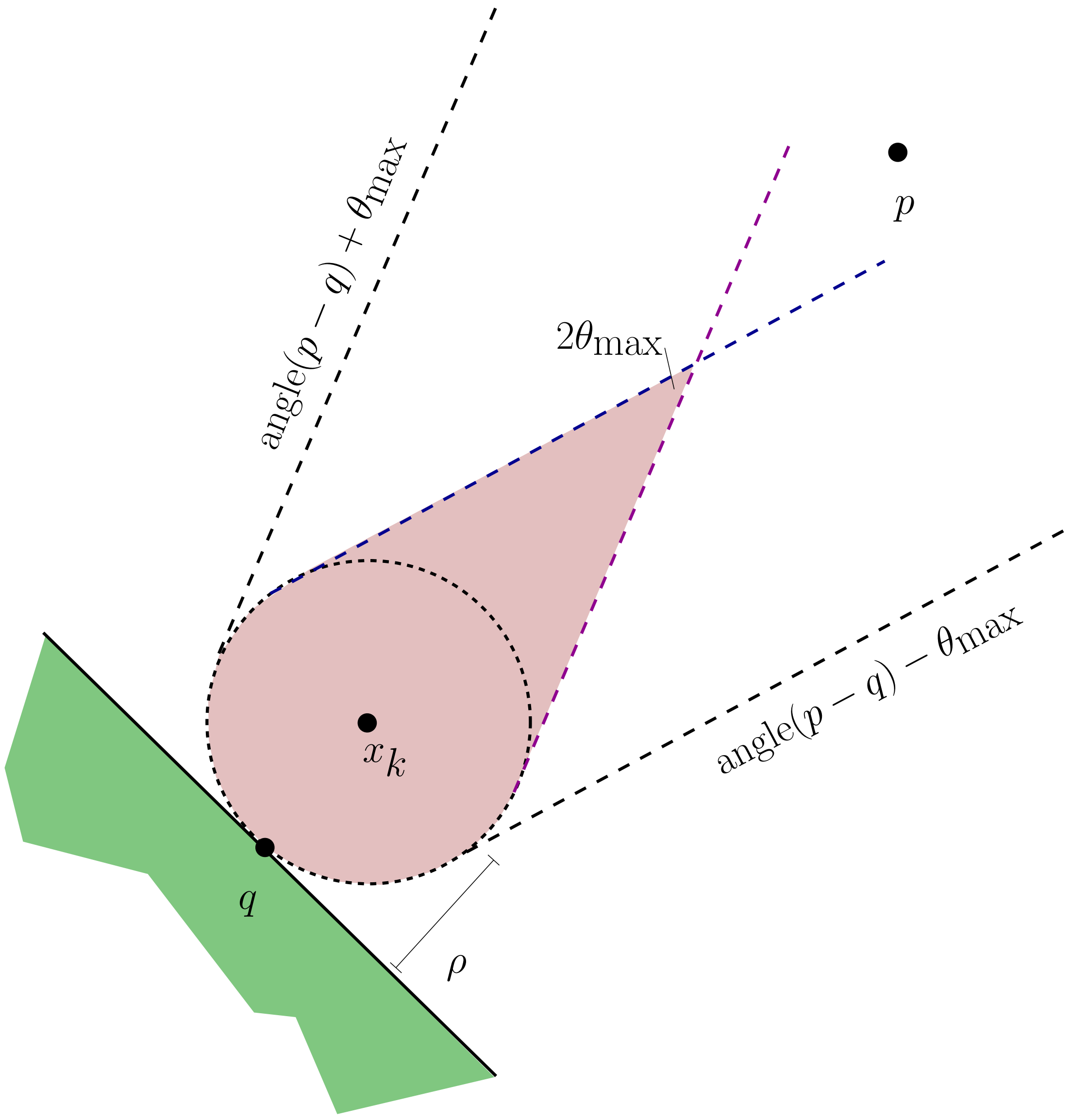

Figure 5.4 Point p is too far from the boundary to be certainly covered by any plan under our robot model. See Theorem 3.

5.4 Algorithm Description

In this section, we describe a method to maximize Area(CCR($u_1, \ldots, u_K$)). The method takes into account the uncertain nature of the robot model's motions and constructs a plan which covers the environment while maintaining a set of states known to contain the robot's true state

The approach is divided into two parts. The first generates the graph, generating parameter-described layers of line segments on the boundary of W as nodes (Algorithm 3), and then adding edges where there are safe actions between segments. The second generates the actual action sequence, by determining which edges in this graph may be traversed in a cycle (Algorithm 5).

Generating the Graph

We define an edge of W in the usual manner, as one edge of a doubly-connected edge list (DCEL). One set of edges represents the boundary of free space, the other set representing the boundary of obstacle space. A face is the line segment between and including the two vertices $\overline{c_i c_{i+1}}$ where i is the index of an environment vertex. Vertices are ordered such that for an edge representing the boundary of free space, a clockwise rotation from $c_{i+1} - c_i$ would be into free space. Note that we do not directly represent both sets, only the set representing the edge of free space.

Our method of creating graph nodes was devised by Daniel Feshbach in [70] and begins by generating the line segments for graph nodes. It creates 'layers' (sets) of segments all of a given length l (Algorithm 3). An illustration is provided in Figure 5.6. The segments may overlap, and are placed evenly along each sufficiently long (at least as long as l) face of W, with the offset between segment starts (and thus amount of overlap) based on parameter $o_{\max}$. The face is filled from one end to the other with segments of length l, start points spaced $o \leq o_{\max}$ apart, until the final

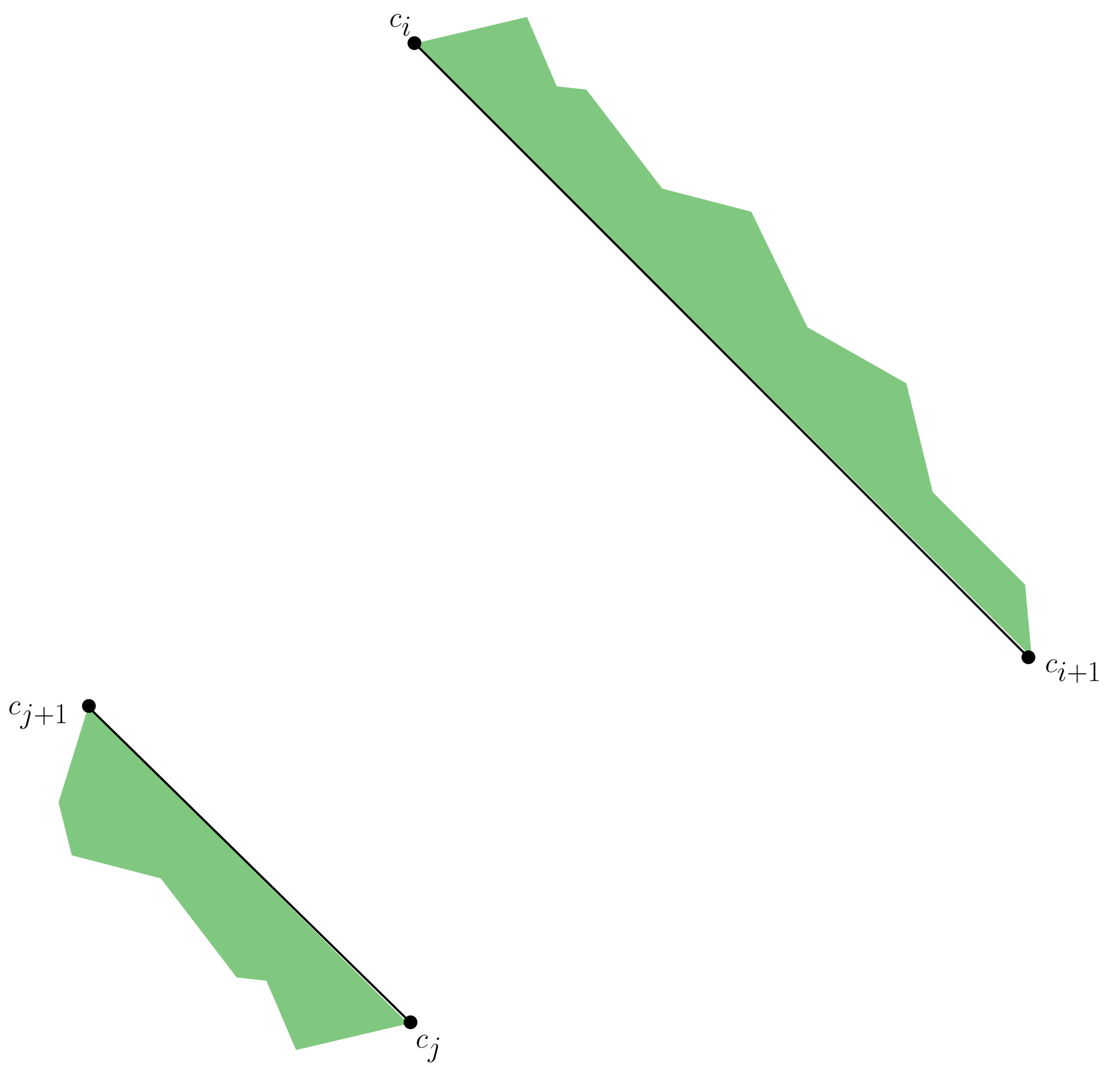

Figure 5.5 Two faces of W, depicting the ordering of vertices along the DCEL edges separating free space from obstacle space.

segment ends at the endpoint of the face. The idea of how o is calculated is to fill the face with segments $o_{\max}$ apart until one includes the end of the face, then move the segments closer together (preserving uniform spacing) until the final segment ends exactly on the end of the face.

Specifically, for each face $\overline{c_i c_{i+1}}$ of W where $l_f = ||\overline{c_i c_{i+1}}||$ and $l_f \geq l$ and a segment $\overline{x_{n_1} x_{n_2}}$, oriented such that the start of the segment is closer to the start of the face and the end of the segment is closer to the end of the face, i.e., $||\overline{c_1 x_{n_1}}|| < ||\overline{c_1 x_{n_2}}||$ and $||\overline{c_2 x_{n_2}}|| < ||\overline{c_2 x_{n_1}}||$, it adds segments $\overline{x_{n_1} x_{n_2}}$, from $n = 0$ until some $x_{n_2} = c_{i+1}$.

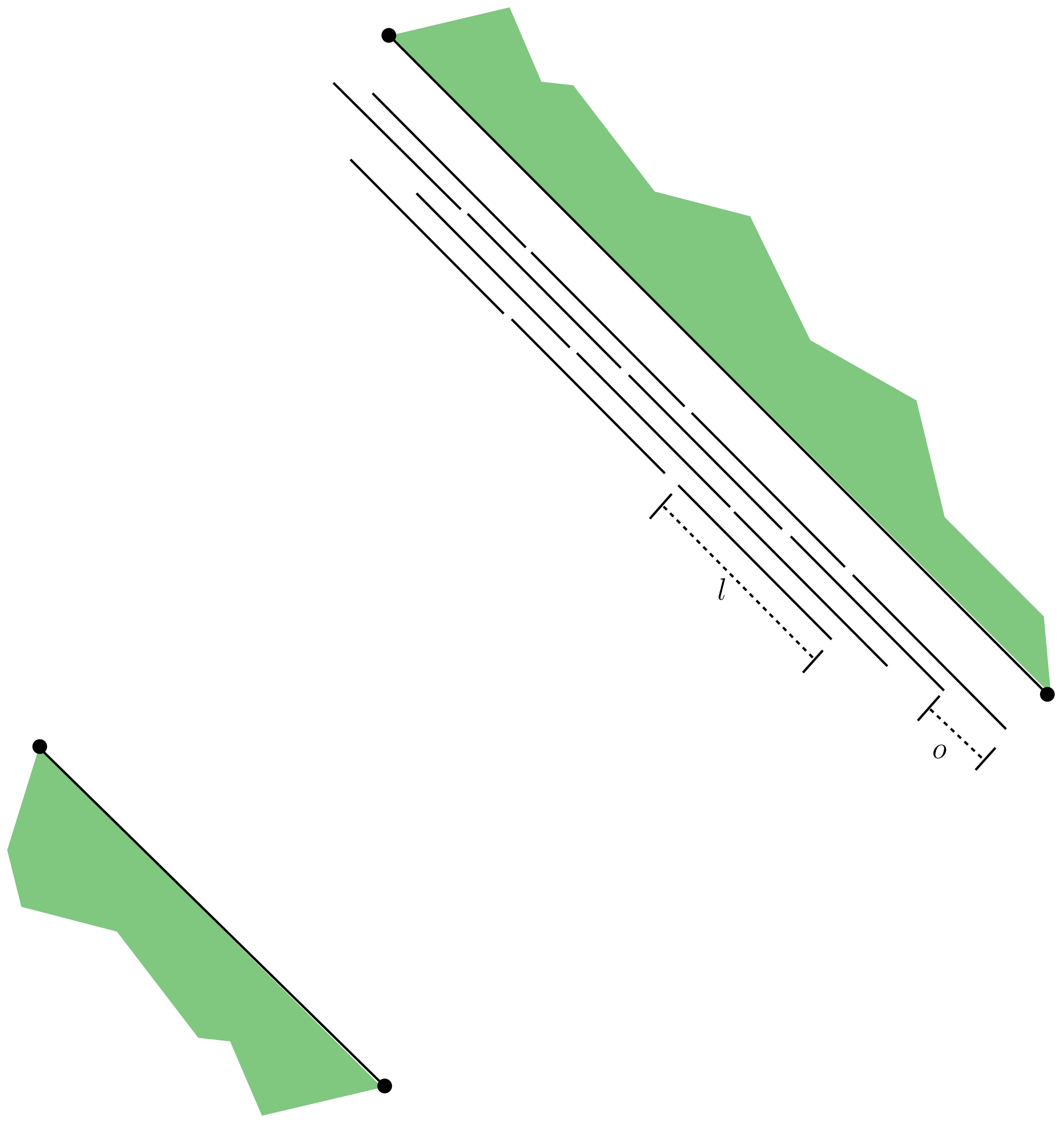

Figure 5.6 A face of W showing a layer of nodes generated by Algorithm 3 with a given length l, and separation o

Algorithm 3 $\textsc{AddLayer}(W, G, l, o_{\max}, z)$

```
1: for face $\overline{c_1c_2}$ of W do
2:   if $||\overline{c_1c_2}|| = l$ then
3:     Add segment $\overline{c_1c_2}$ as a node of $G$
4:   end if
5:   if $||\overline{c_1c_2}|| > l$ then
6:     $o \leftarrow \frac{||\overline{c_1c_2}||-l}{\lceil(||\overline{c_1c_2}||-l)/o_{\max}\rceil}$
7:     $x_1 \leftarrow c_1$, $x_2 \leftarrow c_1$
8:     while $x_2 \neq c_2$ do
9:       $x_2 \leftarrow x_1$ translated $l$ along $\overline{c_1c_2}$
10:      if $z$ is not well defined or $||\overline{c_1x_{i1}}|| \leq z$ or $||\overline{c_2x_{i2}}|| \leq z$ then
11:        Add segment $\overline{x_1x_2}$ as a node of $G$
12:      end if
13:      $x_1 \leftarrow x_1$ translated $o$ along $\overline{c_1c_2}$
14:    end while
15:  end if
16: end for
```

The distance o, then, is $\frac{l_f-l}{\lceil(l_f-l)/o_{\max}\rceil}$.

An additional parameter z is optionally defined for some layers as a limit on how far segments in this layer can be from corners. When defined, it only adds segments which contain some point at most z from either end of the face. Such a layer filters the generated segments to only include those with an endpoint at most z from a corner of the face, i.e., where $||\overline{c_1x_{n1}}|| \leq z$ or $||\overline{c_2x_{n2}}|| \leq z$.

Algorithm 3 is called several times with different values for l, $o_{\max}$, and z, to build up several distinct layers of segments. In this way, we can make the claim that if a path exists for our coverage method, we will find it. However, we cannot claim that when a coverage path does not exist we will terminate. In practice, a halting condition must be established.

After the segments are generated and added as nodes to the graph, edges are found by looping through ordered pairs of nodes and adding them where appropriate.

A directed edge exists between nodes n_i and n_j if there exists an action u_{ij} under which the robot can be guaranteed a safe translation from n_i to n_j. To determine whether such an action exists, it must be true that for any point along the starting node n_i, representing a source segment $\overline{s_1 s_2}$, the action uij is safe to execute and will arrive at the target node n_j, representing a target segment $t_1 t_2$. Algorithm 4 determines whether this edge exits.

The algorithm begins by calculating a range of actions $[d_1, d_2]$, any of which, when executed would result in arrival on the target node from any point along the source node if there are no intervening environmental obstacles. The range is calculated as shown in Figure 5.7. To assure the range is safe, it must be strictly less than π. To be larger means that either the source's face is "away" from the target's or that the target node is too small to allow for $\theta_{\max}$ error.

Next, it must be true that no obstacles exist between the nodes into which actions from the range might carry the robot. Determining this requires two steps: checking the boundary actions and checking the interior actions. Given that there are an infinite number of actions in the range, we cannot check them all in finite space and time. Instead, we observe that after checking the boundary conditions, we can instead consider the finite points in the environment.

On lines 6 and 7, we establish that there is line of sight between the respective ends of the source and target. This step is done by shoot ray, a well-known algorithm running in $O(n\log_2(n))$ where n is the number of vertices in W [10, 49, 92]. If the two points returned are on n_j, then any obstacles blocking translation must be completely contained in the area between the two nodes. The algorithm finishes with the quadrilateral described and a final pass over the vertices of W.

Each edge is labeled with this u_{ij} and the region the action certainly covers as calculated with Theorem 2. It is worth noting that in some rare instances of small source nodes connecting to large target nodes, separated with very small obstacles, there might actually be cases where a disjoint range of actions is safe. That logic is considered by Lewis and O'Kane [72], but not used here. Our method would not create an edge.

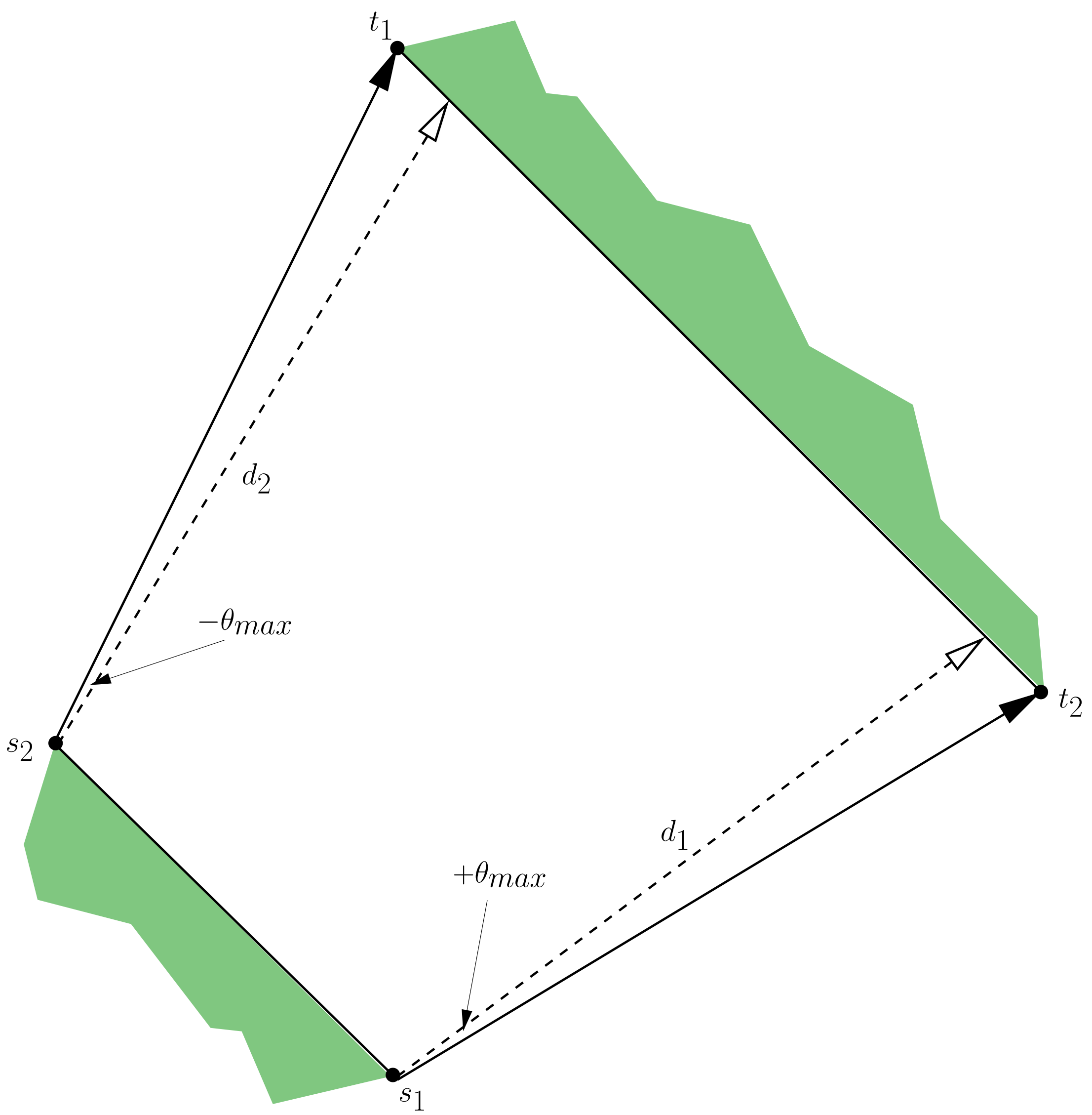

Figure 5.7 Given a source node $\overline{s_1 s_2}$ and target node $\overline{t_1 t_2}$ in W and $\theta_{\max}$, Algorithm 4 determines that there exists at least one safe action between the nodes—d_{mid}, the mid-angle bisector of d_1 and d_2.

Algorithm 4 $\underline{\textsc{HasEdge}(W, \overline{s_1 s_2}, \overline{t_1 t_2}, \theta_{max})}$

```
1:  $d_1 \leftarrow t_2 - s_1 + \theta_{\max}$
2:  $d_2 \leftarrow t_1 - s_2 - \theta_{\max}$
3:  if $\pi <$ the angle from $d_1$ to $d_2$ then
4:      return $\emptyset$
5:  end if
6:  $p_1 \leftarrow$ point returned by shooting ray in $W$ from $s_1$ along $t_2 - s_1$
7:  $p_2 \leftarrow$ point returned by shooting ray in $W$ from $s_2$ along $t_1 - s_2$
8:  if $p_1$ or $p_2$ are not on $\overline{t_1 t_2}$ then
9:      return $\emptyset$
10: end if
11: $q \leftarrow$ quadrilateral formed by $s_1, s_2, p_2, p_1$
12: for $c_i$ in $W$ do
13:     if $q$ contains $c_i$ then
14:         return $\emptyset$
15:     end if
16: end for
17: return the mid-angle bisector of $d_1$ to $d_2$
```

Building The Coverage Plan

After the graph is generated, we have a collection of edges, each labeled with a region of the environment that would be covered if the robot was to cross that edge. It might tempting to simply find edges that are reachable from the start position and greedily attempt to cross those edges representing the most coverage. The approach is problematic because the graph may not be strongly connected due to the underlying navigation method and limited robot model. Selecting a path that crosses one edge, without regard for the forward connectivity of the resulting node to other locations, may leave the robot stuck in a portion of the graph from which it cannot escape to cover elsewhere. One of the largest challenges for our algorithm, for example, is a common feature—a hole in a wall with no nearby faces as shown in Figure 5.8. In this example environment, if the planner leaves the left-most section of the free space before it is covered, it likely will not be able to return without a small $\theta_{\max}$ and very fine discretization of the faces of W.

As a result, our approach to generating coverage plans is based on generating a

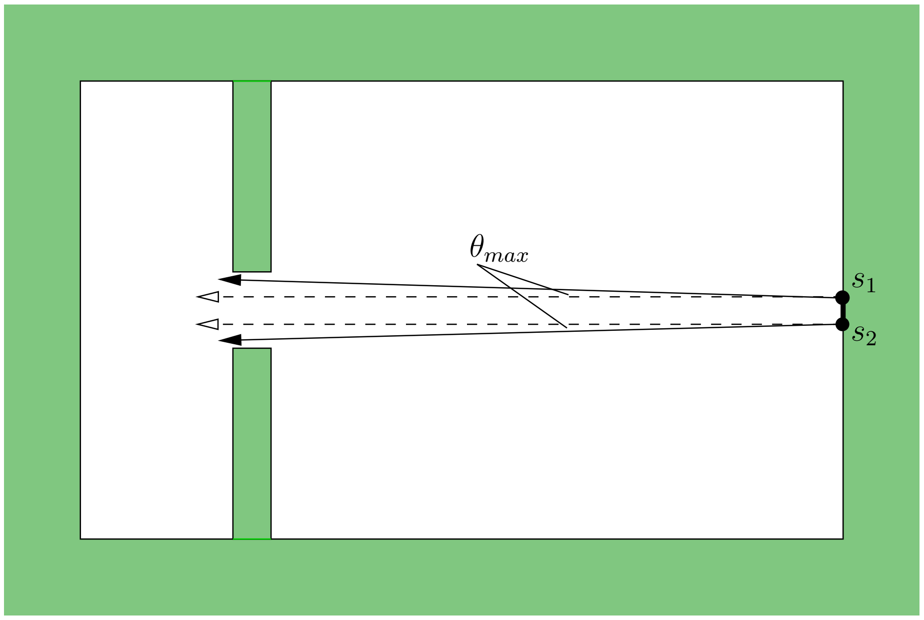

Figure 5.8   A feature common to many indoor environments through which our underlying navigation method has difficulty planning due to the limitations of the robot model. To plan a path through the opening requires some ratio of uncertainty in state to $\theta_{\max}$ and for sufficiently large values of $\theta_{\max}$, there is no state into which the planner can place the robot—for instance, l meters away from a wall.

series of cyclical 'forays' from a node containing the start position, out through the environment to cover some new territory, and then back to the start node. To begin, we first calculate the shortest paths between all pairs of nodes, using the Floyd-Warshall algorithm [43]. The resulting shortest path matrix has enough information to efficiently determine, for any ordered pair of nodes in the graph, whether a directed path exists from the first node to the second node.

We then iterate over the edges of the graph, maintaining a sequence of actions $u_1, \ldots, u_k$ planned to execute, along with $\text{CCR}(u_1, \ldots, u_k)$. For each each e, we check three properties:

1. Is the source node of e reachable from the start node?

2. Is e labeled with a non-empty certainly covered region, which is not already contained in the current CCR?

3. Is the start node reachable from the end node of e?

If all three properties hold, then e represents an opportunity to cover some new portion of the environment. In that case, we generate (using the Floyd-Warshall matrix to determine which states to visit) actions that transit from the start node, across e, and back to the start. For each of the edges crossed by these actions, we include the corresponding certainly covered region in the overall CCR, and remove them from consideration in the outer loop. (Note that some of these edges may be labeled with empty coverage regions, for example because they correspond to segments of uncertainty that are too large. This phenomenon explains why the final CCR produced by the algorithm need not be a connected set, e.g. Figure 5.11)

After each edge has been considered, the planning process terminates. The results is a sequence of actions —the coverage plan itself— that crosses every edge that can be crossed without becoming trapped away from the start vertex, along with the CCR corresponding to that coverage plan.

5.5 Experiment Results

We implemented our algorithm using C++ and OpenGL. We simulated a robot with $r = 0.3$, and the layers of segments specified in Table 5.1 as our graph nodes. We scaled the layer parameters and robot size relative to the size of the environment.

In an effort to characterize the performance of our algorithm as error grows, we ran the coverage experiments several times in each environment, increasing $\theta_{\max}$ in each iteration. In all executions the robot's initial position is the same and affects the coverage possible for an environment. We used a heat map to illustrate the portions of the environment that are no longer coverable, reachable, or from which the robot can no longer guarantee a reliable return.

We selected four environments. Figure 5.9 is a maze-like environment represent-

Algorithm 5 $\textsc{ComputeCoverageEdges}(\overline{pq}, \text{CCR}, G)$

```
1: $P \leftarrow \textsc{AllPairsShortestPath}(G)$
2: $\textsc{cover} \leftarrow$ empty set of edges
3: for all $e \in$ edges of $G$ do
4:   $S \leftarrow P[\overline{pq}][\textsc{source}[e]]$
5:   $T \leftarrow P[\textsc{target}[e]][\overline{pq}]$
6:   $\text{CCR}' \leftarrow \textsc{ComputeCoverage}(e)$
7:   if $S \neq \emptyset$ and $T \neq \emptyset$ and $\text{CCR}' \setminus \text{CCR} \neq \emptyset$ then
8:     for all $s \in S$ do
9:       $\text{CCR}'_s \leftarrow \textsc{ComputeCoverage}(s)$
10:      $\text{CCR} \leftarrow \text{CCR} \cup \text{CCR}'_s$
11:      $\textsc{cover} \leftarrow \textsc{cover} \bigcup s$
12:    end for
13:    $\text{CCR} \leftarrow \text{CCR} \cup \text{CCR}'$
14:    $\textsc{cover} \leftarrow \textsc{cover} \bigcup e$
15:    for all $t \in T$ do
16:      $\text{CCR}\,'_t \leftarrow \textsc{ComputeCoverage}(t)$
17:      $\text{CCR} \leftarrow \text{CCR} \cup \text{CCR}'_t$
18:      $\textsc{cover} \leftarrow \textsc{cover} \bigcup t$
19:    end for
20:  end if
21: end for
22: return $\textsc{cover}$
```

Table 5.1 Layers of Segments Used in Simulation

l (length)	$o_{\max}$ (offset)	z
3	2	
2	1	
1.5	1	
1	0.5	
0.5	0.375	2
0.25	0.1	0.3

ing a building or office space. This map illustrates some of the more pronounced difficulties plaguing the underlying navigation method. The cyan-shaded coverage region demonstrates the planner's inability to find a path into the T-junction near the middle of the lower hallway at that error bound. Note that as error grew, the planner quickly lost the ability to reliably move from the inner portion to the outer hallway and then could no longer reliably cross from the upper section into the lower.

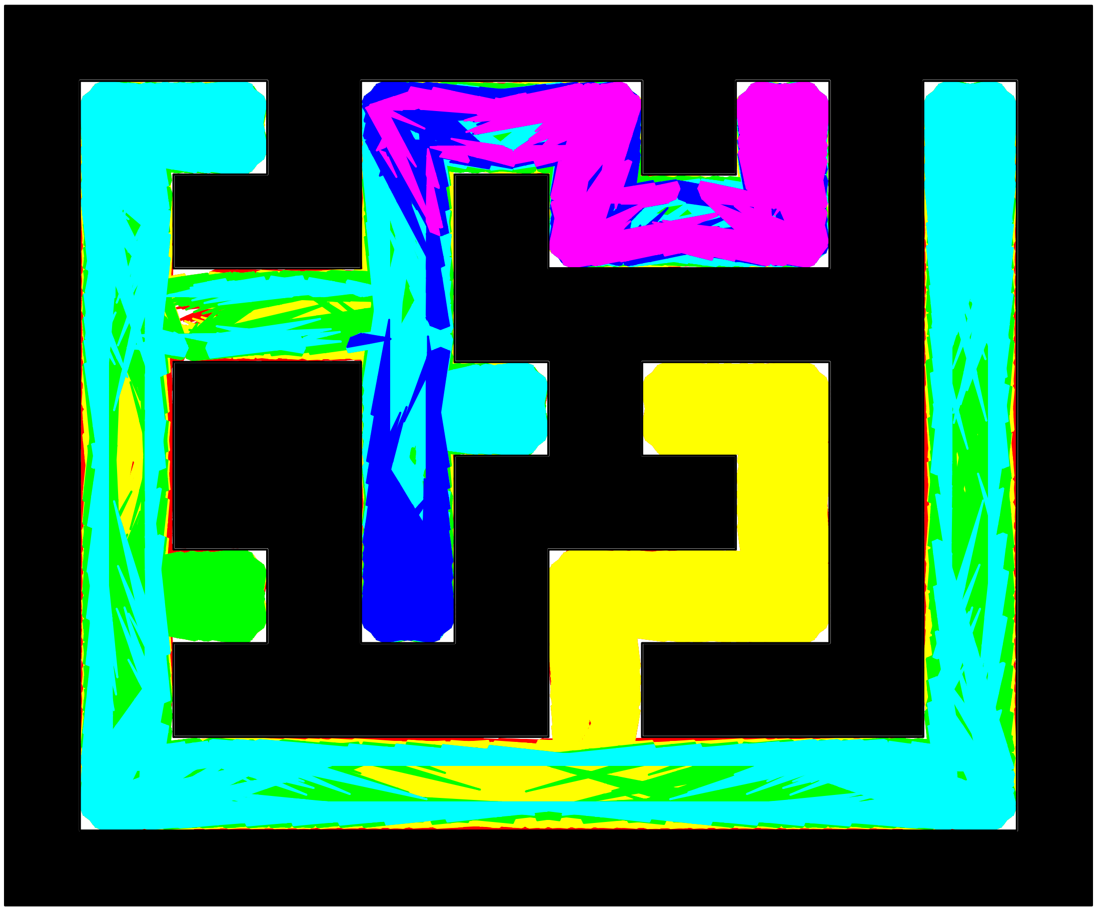

Figure 5.9 An office-like environment presenting both challenges in navigation and coverage. The robot begins in the third convex vertex from the right in the top-most section of the map and loses the ability to connect its starting nodes with the rest of the environment nodes when $\theta_{\max}$ is ± 3.

.

Note also, that for all $\theta_{\max}$, each heat map covers nearly the entire area reachable by the navigation algorithm. In an environment like this, the coverage is predictable because no point is distant from an environment edge. The missed coverage in the left vertical hallway and bottom hallway are, again, due to difficulties in navigation planning. The openings represent a difficulty in finding a small (localized) node to which to navigate. Without the localization, coverage may occur, but cannot be guaranteed.

Figure 5.10 is a large mostly empty environment to illustrate points far from any edge which the robot cannot be guaranteed to cover. This environment is interesting because while it is mostly open and therefore easy for the underlying planner to navigate the obstacles, coverage is difficult for the same reason. Even though the robot is able to navigate to every edge of the environment for every value of $\theta_{\max}$ save the last two, the rate at which coverage drops is dramatic. This occurs in the "shadow" created by the larger corner obstacles. In the work by Erickson et al. [80], it was discovered that perpendicular edges forming a convex vertex are useful in reducing uncertainty by driving it toward a single point—the convex vertex. Though the behavior is not encoded in the planner, it is used and is inhibited by the obstacles. We will later determine that for all but the largest values of $\theta_{\max}$ it maintains the lowest percent coverage of any of our environments, despite the ease of navigation.

Figure 5.11 is a simple environment with two large holes separating the convex vertices of the environment. The environment is symmetric across both axes and so is least affected by the robot's starting point. However, even this simple environment illustrates the two main stumbling points of our navigation method—the long halls along the top and bottom and the T-junction in those halls. Coverage also becomes difficult at those points, due to the error accumulated through the necessary translation back-and-forth between the walls. At the point at which the robot is able to translate into the vertical hallway, it has uncertainty such that it is unable to guarantee coverage of the opening. Using a version allowing a human to "drive" the robot, coverage could be achieved. The method is further discussed in Section 6.2.

The final environment we present in Figure 5.12 demonstrates the algorithm in a more natural environment. There were fewer collinear points and less rectilinear environment faces. Our planner did very well in this environment, both in terms of navigation and coverage. While there are openings large enough to allow translation, there are also many environment faces to provide temporary localization. Even at the highest level of uncertainty, the planner was able to provide a path visiting every

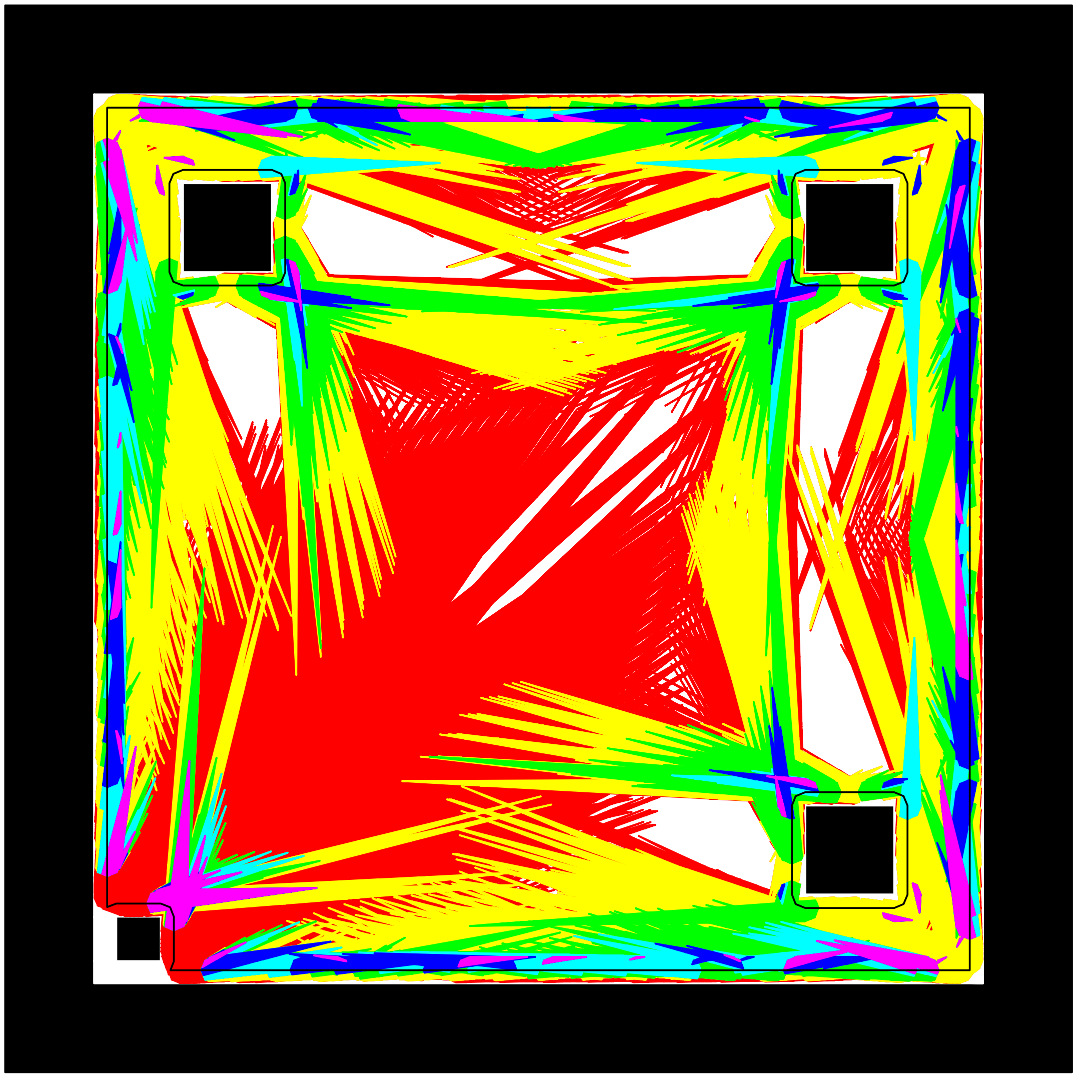

Figure 5.10 An environment with a relatively large empty region as well as obstacles inhibiting our algorithm's localization method.

convex vertex of the environment and was only unable to guarantee coverage of those portions of the environment distant from faces.

In all cases, save the environment in Figure 5.10, the algorithm achieved close to 100% coverage at $\theta_{\max} = \pm 0.5$ degrees.

To compare performance in the different environments, we calculated, for each $\theta_{\max}$, the volume of W contained in the certainly covered region. The least structured environment from Figure 5.12 was clearly best handled by the algorithm. Whether

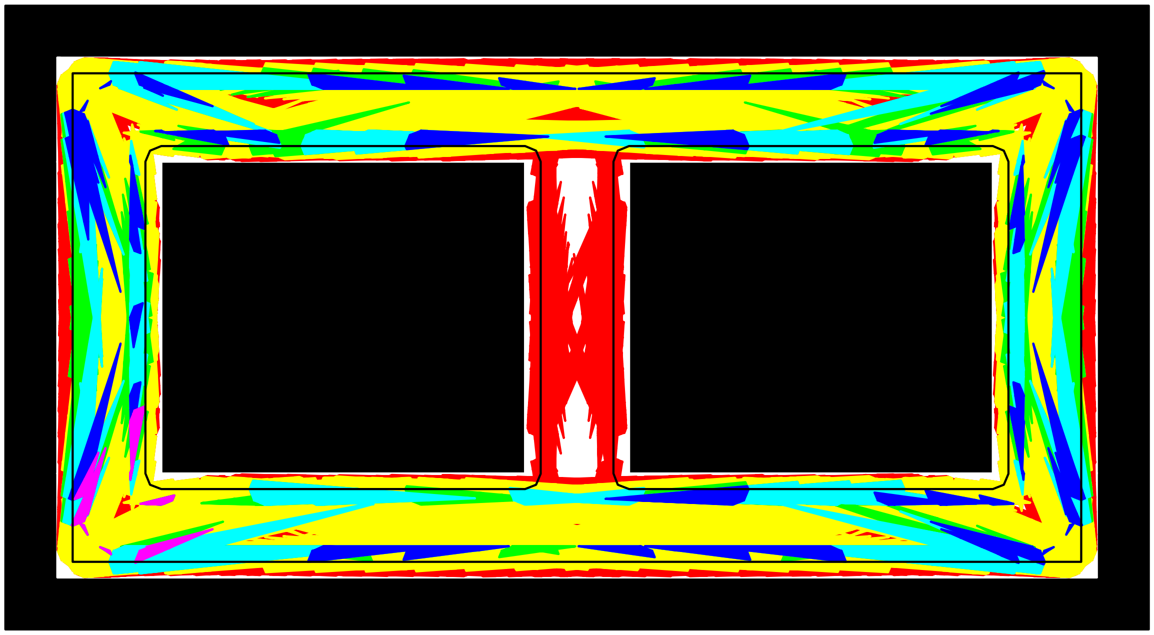

Figure 5.11 A simple environment with two large holes. The robot began in the lower left corner and became stuck there as $\theta_{\max}$ reached ± 3 degrees.

this is a function of the decomposition method or the method itself is not apparent. Not surprising, the environment for which our algorithm performs worst is the mostly empty Figure 5.10. This environment's obstacle-blocked convex vertices and largely empty center is ideally suited to thwart our planner. On the graph of our planner's coverage Figure 5.9, the point at which there is no path back into inner portion of the map is clearly visible at $\theta_{\max} = \pm 2.5°$. The brick-shaped environment in Figure 5.11 has a fairly shallow loss of coverage except for the point at which the center hallway is lost—when $\theta_{\max}$ reaches $\pm 1.5°$.

These results are not unexpected due to the navigation method. Even with many more data points the behavior likely remain a gradual loss of coverage as error increases until some section of the environment is lost and a dramatic reduction is seen in coverage.

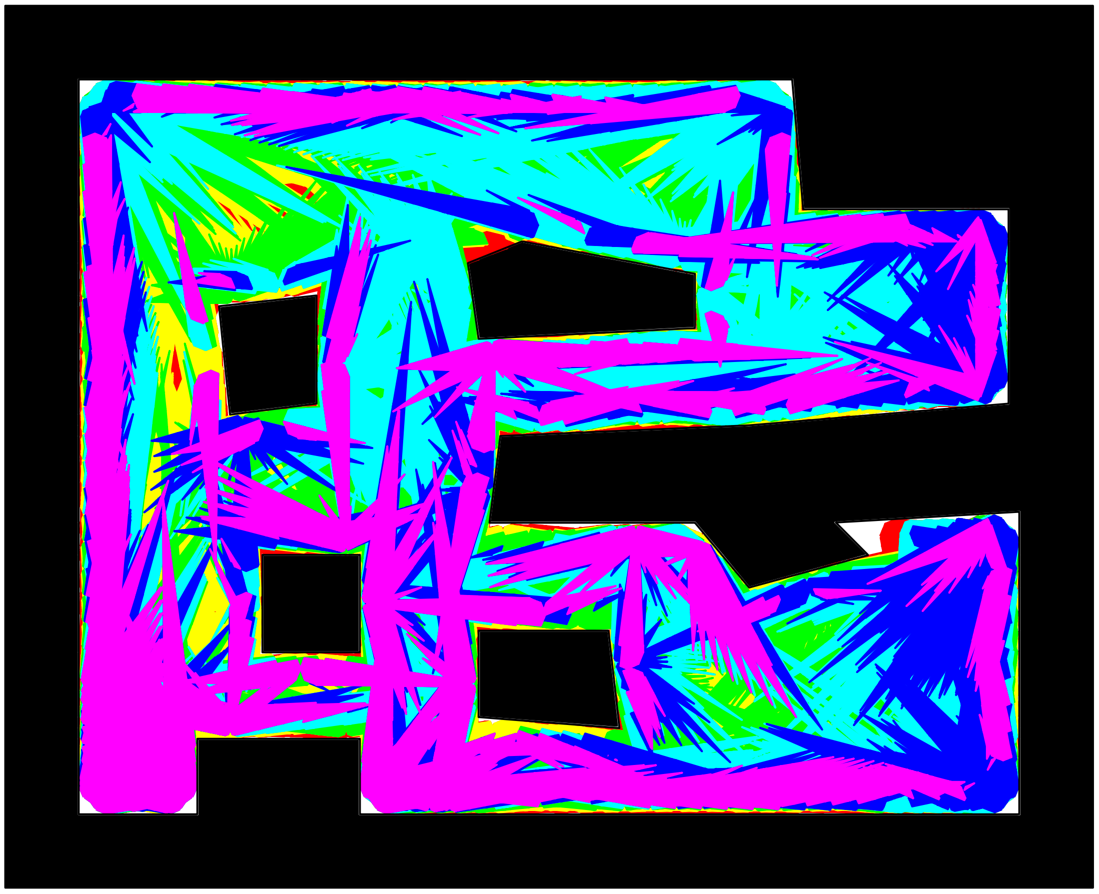

Figure 5.12 A more natural cave-like environment. As error grew, the robot retained the ability to navigate around most of the environment, but not the ability to cover the more spacious open areas. This environment demonstrates the methods ability to deal with non-uniform features.

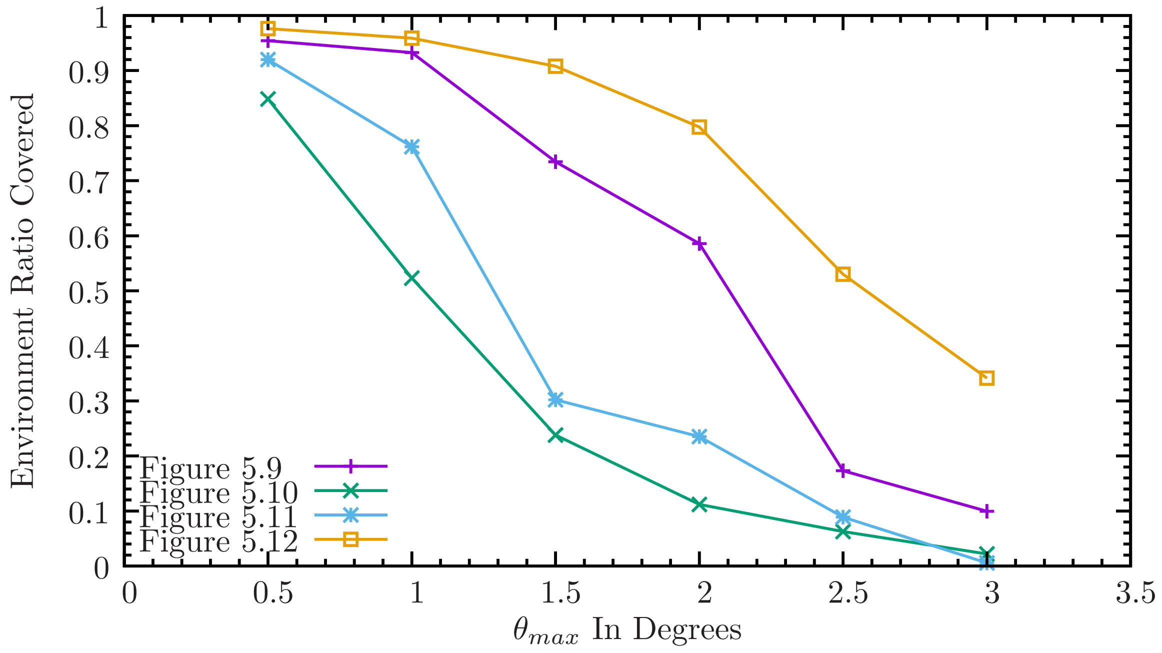

Figure 5.13 A plot comparing the performance of the algorithm in the four different environments when accuracy degrades as $\theta_{\max}$ is allowed to grow.

Chapter 6

Conclusion

In both our Dubins coverage and blind coverage algorithms, the critical decision-making begins with the decomposition. In Dubins coverage, we sought some minimal amount of coverage, while in blind coverage we sought states which resulted in the most coverage possible by a single action; other states were just in place to translate to those states allowing coverage. The difference between the two problems is one of state estimation. In the Dubins coverage model, the robot may always determine its location based on an oracle—a sensor which can answer some question with veracity. The question, obviously, being "what is my state?" The robot model used in blind coverage needs to keep some idea of state based on previous actions. This is called the robot's information state (I-state).

The need to calculate and maintain an I-state determines the sort of discretization into which we can decompose the environment. We need states *large* enough to contain accumulated error represented by an I-state. Because there optimality is not a requirement, do not seek to provide the algorithm with more states than what are required to provide coverage. Our goal is simply to provide enough states to allow the planner a chance at complete coverage. From this point of view, our goal with Dubins coverage is to provide every possible useful state for the robot and let the planner choose which set of states results in an optimal circuit. As stated in Chapter 1, we consider the problem from two different angles—first, given complete and constant knowledge of state can we determine the optimal path for coverage, second, what does a coverage plan look like when the translation problem must be solved simultaneously?

6.1 Dubins Coverage

Our stated goal in Dubins coverage is, given perfect knowledge of state, calculate the shortest possible path to completely cover a given environment. We proved that an optimal coverage path will require the solution to an NP-Complete problem running in $O(2^n\sqrt{n})$ time [13,54], where n is the number of discrete locations a robot must visit to ensure complete coverage. This implies an algorithm very sensitive to environment discretization. The reason the algorithm in [107] is able to run in polynomial time is due to trade offs made in their very clever discretization.

Rather than mapping their graph vertices to some unit of coverage and graph edges to the non-coverage translation between states, the authors mapped coverage (via cells) to graph edges and states to the critical points which created the coverage cells. They were careful to ensure that all edges were required and that a circuit existed without covering a cell multiple times. That is, they constructed the graph to contain an Eulerian circuit [15]. This allowed them to solve their problem as an instance of the Chinese postman problem (CPP), running in polynomial time [34]. The CPP algorithm is stated as: given a graph, determine the shortest circuit which visits every graph edge at least once. The solution to CPP is an Eulerian tour.

Open problems

Decomposition: An observation made as a result this study is that even with perfect knowledge of state, coverage is still a hard problem. When the navigation problem is solved, if one requires an optimal plan for coverage, there still exists a very difficult problem. In our decomposition, we made certain assumptions about the environment—there is a "best" axis with which we could align coverage. In any

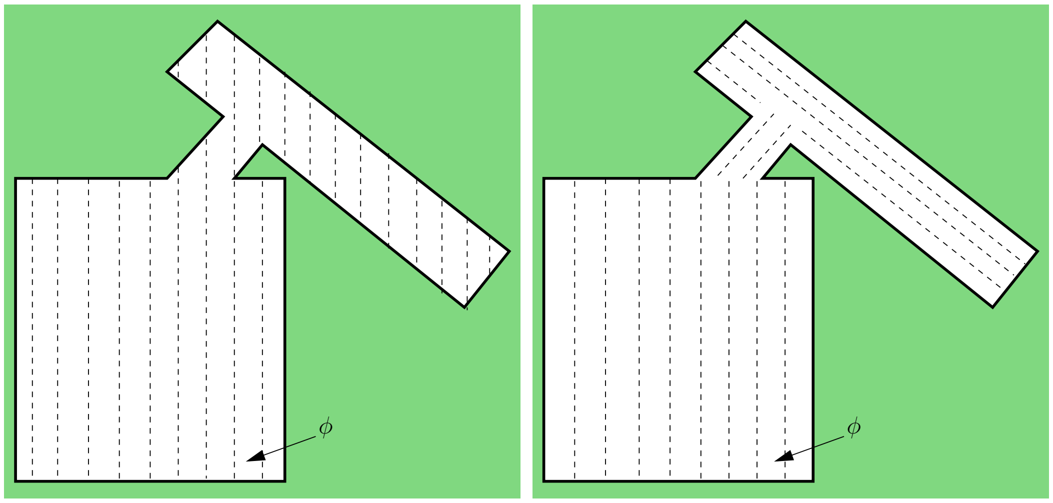

Figure 6.1 (left)An environment for which there is no single axis of coverage by which translations between coverage passes is minimized. (right)A decomposition of the environment such that it is covered along different axes.

unstructured environment, this is unlikely to be the case, such as the Lake Murray example in There is no obvious single axis of coverage. Figure 6.2. This is an important failure by both of our discretizations. The Boustrophedon coverage strategy can be effective because it minimizes time spent not covering the environment. To be most effective, however, it requires passage along the axis aligned with some idea of the length of the free space—the longest chord aligned with some axis. Consider the left image from Figure 6.1.

The left image depicts how our algorithm might decompose the environment to create coverage nodes for our graph. Ideally, a planning algorithm would divide the environment into at least two axes of coverage. One axis would be used to cover the diagonal region at the top, one to cover the x/y-axis-aligned area at the bottom as illustrated in the right image of Figure 6.1. A recent method presented by Brown and Waslander [20] considered a decomposition based on "growing" the environment edges to constrict the environment. This approach likely holds a good deal of promise in the decomposition. Brown and Waslander use their constriction method to create a single path of coverage, by blending the cells into one large continuous cell. It would

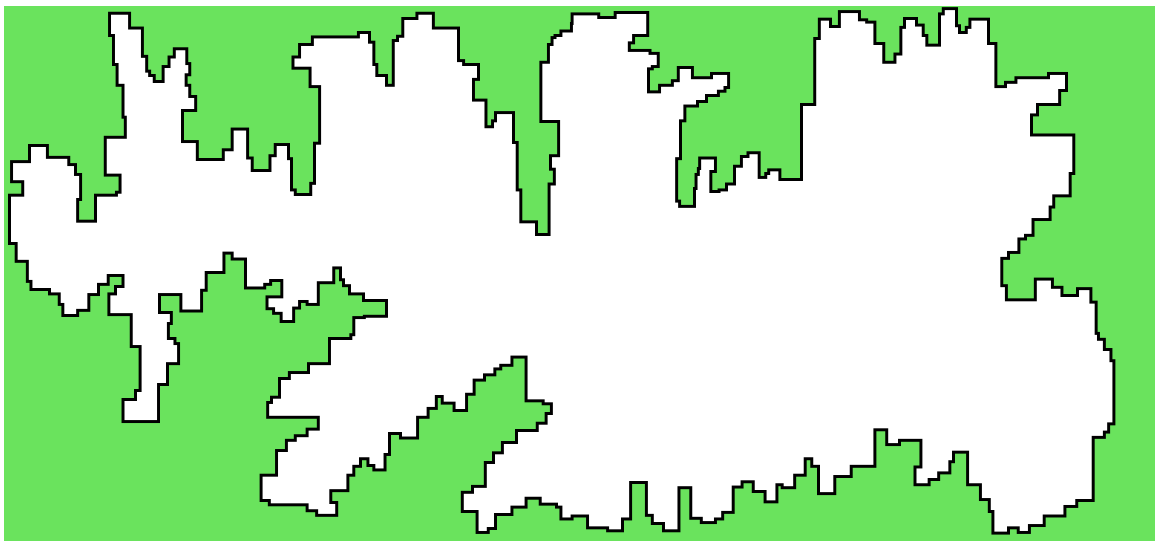

Figure 6.2 A geometric representation created from a satellite map of Lake Murray near Columbia, SC.

be worthwhile to try leaving them as cells and allowing our planner to attempt to order passage. This is necessary because our model has non-holonomic constraints.

Dubins path planning with obstacles: In our decomposition, we avoid the introduction of environment obstacles. In the case of airborne robots this is not a bad assumption with sufficient altitude. It is equally true with sufficient depth and a robot boat. However there exist many cases in which some idea of obstacles is necessary. This seems like a simple extension at first: devise a method to determine a path between two nodes, if a path is found add an edge, profit [84]. However our decomposition may be unable to find edges when obstacle space shares a border with space which must be covered. If they share a border and the decomposition is such that a vertex begins or ends at obstacle space, then there will be no safe action entering or exiting the vertex, respectively.

The standard method for dealing with obstacles like this would have us use a Minkowski sum of the obstacle with a vector of magnitude ρ. This will provide the necessary space to allow coverage of each obstacle, but will incorrectly remove areas

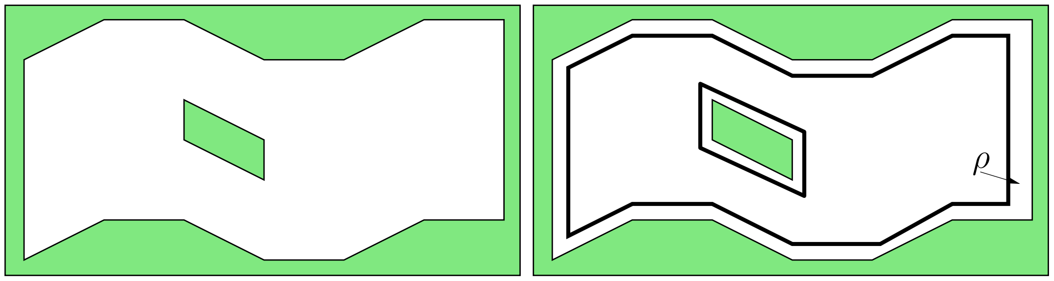

Figure 6.3 Creating a Minkowski sum from the obstacle edges of the environment and a vector of magnitude ρ.

the robot could actually cover. Observe in Figure 6.3, if the direction of coverage is the y-axis, then much of the environment is unnecessarily avoided—that is, the outward growth along the x-axis of the obstacle space.

While in our particular case this can be remedied by a custom version of the process avoiding growth perpendicular to the direction of coverage, it becomes interesting to consider this problem if a complete solution is desired. How are the obstacles grown to ensure that navigation through a space can be achieved?

6.2 Blind Coverage

When we began consideration of the problem of blind coverage, we created a tool which allowed a human to "drive" the robot. We found several strategies to further progress of coverage. Our algorithm makes explicit use of no strategy. Its ability to cover the environment depends entirely on the discretization step. The discretization step, unfortunately makes almost no use of domain specific knowledge of the problem. The second thing which becomes obvious when driving the robot is that some edges are worth more than others, but due to navigation and the coverage provided during navigation, it is not clear how to weight edges.

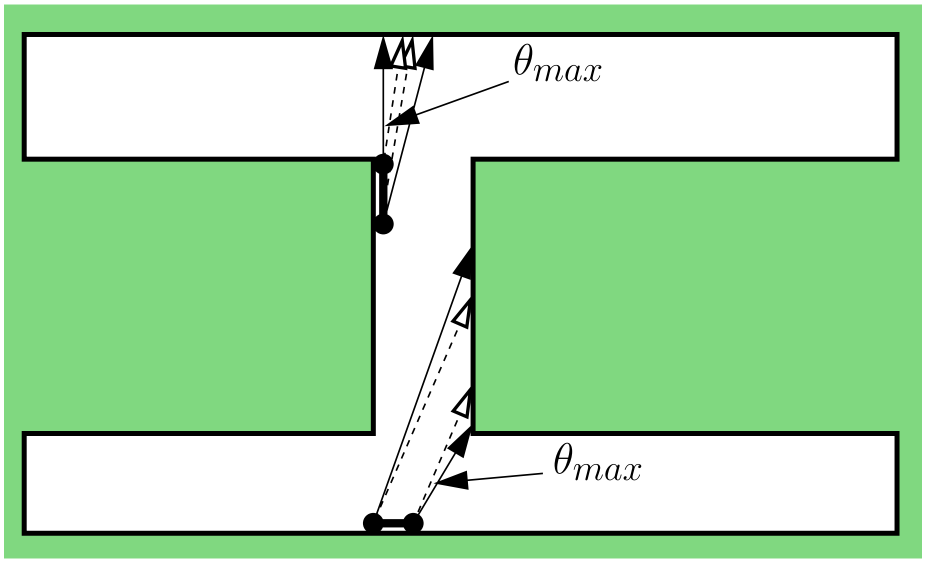

Figure 6.4 A strategy by which a robot might remain localized well enough to guarantee coverage of a T-junction.

Future work

Two explorations which may be useful are the application of a model-aware environment decomposition and a directed path generation algorithm.

Model-aware decomposition: When given the task of driving the model robot in an environment, a human noticed a useful strategy for covering the area of a T-junction. It was obvious that the three perpendicular walls provide a means to remain localized well enough to cover the area. Figure 6.4 illustrates a strategy whereby a planner may localize the robot using the perpendicular walls of the T-junction. The transition across the hall is not shown, but allows the strategy to be mirrored from what is shown. The robot makes sweeps, passing from the inner hallway into the outer and back-again. Our current algorithm makes no use of the relationship between $\theta_{\max}$ and nodes more likely to have translations to and from others.

The decomposition in our blind navigation work [72] used a ray-extension algorithm between sets of "visible" environment vertices to generate sets of points along the edges of the environment in an effort to make use of the relationship between $\theta_{\max}$ and safe translations. For each pair of visible environment vertices $\overline{c_i c_j}$ of the environment, we extended rays from c_i in the directions $c_i - c_j \pm \theta_{\max}$ and c_j in the directions $c_j - c_i \pm \theta_{\max}$. The resulting intersections were then used along with the environment edge vertices to guide a "wallpaper" process similar to this thesis. It would be interesting to both replace and augment the current "wallpaper" strategy. Additionally, though we do not specifically include convex vertices in the nodes, introduction of our corner-finding strategy would allow us to include them and is an obvious next step.

Path generation: We make no claims of optimality nor do we attempt to reduce the length of the paths generated by our planner. This is a two-fold problem. In the first step, one must choose a set of necessary edges which covers the environment and in the second choose an ordered list of edges which includes all those from the necessary set. Our current method takes edges in no particular order and, if the edge provides some new coverage, adds it and the plan to reach it and return to the start node to the path. While it might seem tempting to simply order their selection by the volume of the area of the map covered, this value will change as other edges are selected. Unless every edge is updated each time an edge is selected, the order will quickly grow stale and would offer nothing more than random selection.

An idea which may be worth investigating is keeping some idea of how much edges are adding to coverage. When the "next biggest" edge is selected, the amount of coverage offered is compared to the threshold and if it is currently too small, its coverage (and therefore weight) is updated and it is placed back into the edge collection. While eventually many edges may need be updated, it would supply an idea of what is a "good edge" based on what has been seen. We provide a rough sketch of our idea in Algorithm 6. The algorithm ends when there are no edges

Algorithm 6 $\underline{\text{ComputeCoverEdges2}(\overline{pq}, G)}$

```
 1: CCR ← empty region
 2: UpdateThreshold($\emptyset$, CCR)
 3: threshold ← CalculateThreshold($\emptyset$)
 4: $P$ ← AllPairsShortestPath($G$)
 5: $E$ ← empty set of edges
 6: $Q$ ← MaxHeapify(edges of $G$)
 7: while not Empty($Q$) do
 8:   $e$ ← Remove($Q$)
 9:   $S$ ← $P[\overline{pq}]$[source[$e$]]
10:   $T$ ← $P$[target[$e$]][$\overline{pq}$]
11:   if $S \neq \emptyset$ and $T \neq \emptyset$ and coverage[$e$] $\setminus$ CCR $\neq \emptyset$ then
12:     threshold$'$ ← CalculateThreshold(coverage[$e$])
13:     if threshold$'$ $\leq$ threshold then
14:       $E \leftarrow E \bigcup e$
15:       UpdateThreshold(threshold$'$, threshold)
16:       threshold ← CalculateThreshold(coverage[$e$])
17:     else
18:       coverage[$e$] ← coverage[$e$] $\setminus$ CCR
19:       if coverage[$e$] $\neq \emptyset$ then
20:         Push($e, Q$)
21:       end if
22:     end if
23:   end if
24: end while
25: return E
```

remaining to which the robot can translate which offer additional coverage and the edges have their coverage computed when they are added to the graph. The volume of an edge's coverage area adds to the current CCR is its weight. The mechanism to compute and maintain the threshold is left ambiguous as a correct strategy is not immediately evident. Likely it must keep up with a history of edges' additions to the CCR and offer a comparison of what any new edge contributes as some ratio of additional coverage.

Notice that the output of Algorithm 6 is not a plan for coverage, but a set of edges which results in coverage of the environment. A naive approach might be to solve the problem as a rural Chinese Postman Problem, a known NP-Hard problem [69]. While we have taken no steps to prove any aspect of blind coverage's hardness, given our proof of optimal Dubins coverage, it would not be surprising to discover this problem is at least as hard.

Open problems

Probably the most outstanding question left open by blind coverage is how to optimally discretize the environment. Stated exactly, what sort of discretization results in a graph containing the shortest path to cover the environment? From our previous work, we know that actions between states result in more coverage when the edges from which the action begins and ends are more perpendicular edges. This occurs due to the way error is modeled. Actions perpendicular to an edge result in the largest accumulation of error. The more parallel an action is to the edge, the less error accumulates.

Optimal decomposition: Even though armed with this knowledge, our planner still uses a nearly-generic state generation method. Our first planner was intended to include on the robot's initial position and base future states on that position. The idea was to start at the robot's beginning state and perform a forward search through the environment searching for states. Algorithm 7 provides a rough sketch for a method to generate states from a given starting point along all possible directions, discretized by some δ. The algorithm is sensitive to the parameter because both the number of states generated by the algorithm, as well as the location of the states themselves are dependent on it; the number of states being capped at $\lfloor \frac{2\pi}{\delta} \rfloor$.

The rest of our method is presented in Algorithm 8. Given a starting point, the algorithm begins a loop calling GenerateStates from that starting point which sweeps around the starting point and makes note of every line segment to which the robot could translate with a single action safely. If those line segments are new, then

Algorithm 7 GENERATESTATES($\overline{pq}, W, \theta_{\max}, \delta$)

```
1: $V \leftarrow \{\}$
2: $d \leftarrow 0$
3: while $d < 2\pi$ do
4:     $d_q \leftarrow d + \theta_{\max}$
5:     $t_q \leftarrow$ SHOOTRAY($W, q, d_q$)
6:     $d_p \leftarrow d - \theta_{\max}$
7:     $t_p \leftarrow$ SHOOTRAY($W, p, d_p$)
8:     if HASEDGE($W, \overline{pq}, \overline{t_q t_p}, \theta_{\max}$) then
9:         $V \leftarrow V \bigcup \overline{t_q t_p}$
10:    end if
11:    $d \leftarrow d + \delta$
12: end while
13: return $V$
```

Algorithm 8 FORWARDSEARCH($\overline{pq}, W, \theta_{\max}, \delta$)

```
1: $Q \leftarrow \{\}$
2: PUSH($Q, \overline{pq}$)
3: $V \leftarrow \{\}$
4: while not ISEMPTY($Q$) do
5:     $q \leftarrow$ POP($Q$)
6:     $S \leftarrow$ GENERATESTATES($q, W, \theta_{\max}, \delta$)
7:     for $s \in S$ and $s \notin V$ do
8:         $V \leftarrow V \bigcup s$
9:         PUSH($Q, s$)
10:    end for
11: end while
```

it collects them into the vertices of the graph and considers GENERATESTATES for each. The structure Q is purposely left ambiguous between a queue and a stack. From what limited analysis we performed, there was no appreciable difference between the two. The halting condition for the loop is when no new state locations are discovered. Obviously the process could go on for a very long time depending on the step size of the sweep and the definition of equality. In practice, we did not get good results for this method. It generated far too many nodes. In addition to consuming large amounts of resources to create, the runtime of our coverage algorithm was untenable.

The optimal method to decompose the environment, of course, must be derived from the goal—coverage. In a most basic sense, the environment would be decomposed based on the coverage shape of the robot's actions. Segments or states, would then be created to cover the decomposition. In this formulation, the edges are generated first and then the states which must exist to support the edges generated second. Once the segments are generated, then more would need to be added to allow translation and localization between the source and target nodes for the coverage edges. Generating these without sampling or arbitrary discretization has the trappings of an intractable problem.

Expanding uncertainty Another open question we leave is that of determining the largest amount of uncertainty we can allow in robot state. In our current definition of *safe states* resulting from *safe actions* all states must lie along a single environment edge. We note that some state might span multiple environment edges while still allowing for manageable state estimation and in some cases even allow some guaranteed coverage from those states. Any complete decomposition of the environment for this robot model must allow states such as these to exist. It is currently unclear what changes are necessary for tests to ensure that localization can be maintained to and from states such as this. Likely there would be several new approaches required to test for different the different sorts of edge-spanning nodes that could result. The two main divisions of spanning nodes would be nodes which span two edges which share a vertex and spanning nodes which span two edges that do not share a vertex. In the case of edges which share a vertex, there are only two options, those which span a convex vertex as illustrated in Figure 6.5 and nodes which span a reflex vertex. The other class of spanning nodes offers much more variation because they need not be close. An error cone which projects over a hole, for instance might span two edges which are very close to one another in two axes along with a third edge far from the first two.

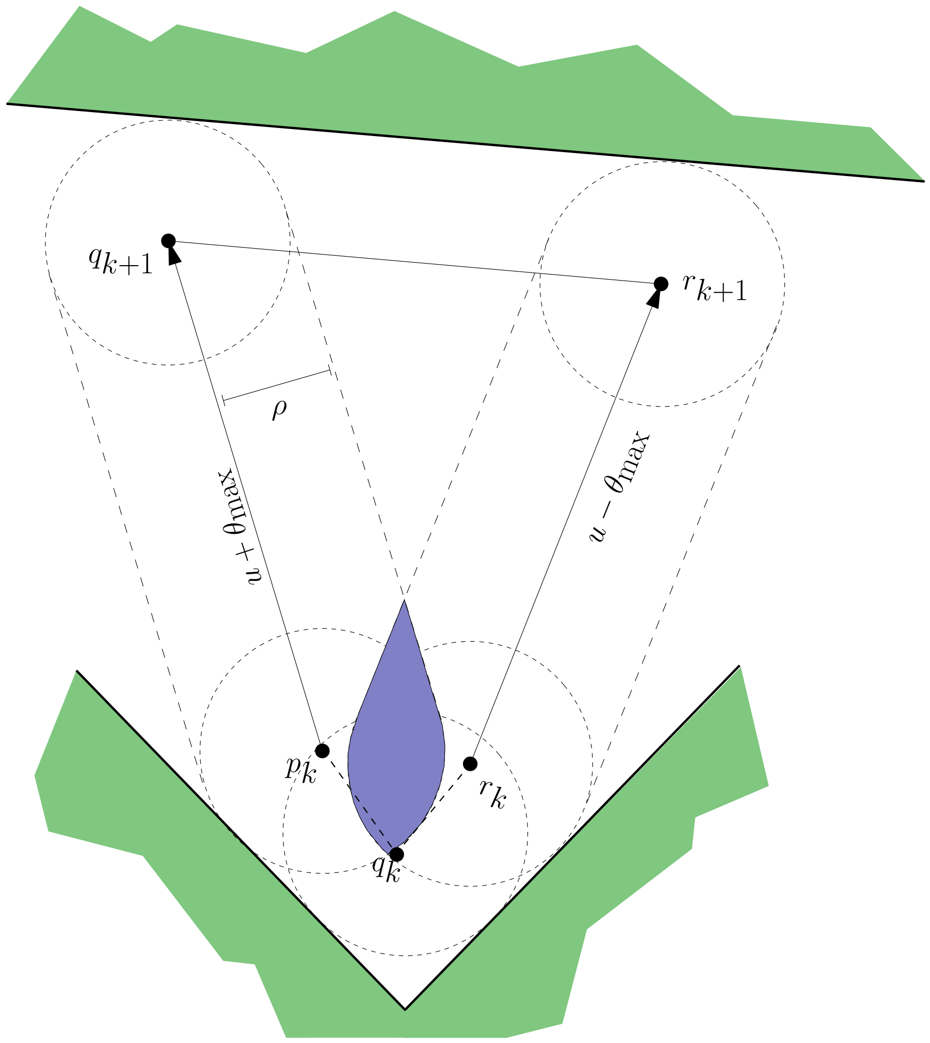

Figure 6.5 Computing the CCR for a single action from a state spanning more than one environment edge.

Max($\theta_{\max}$) The last open question we leave is that of determining a correlation or calculation from the largest amount of error our robot model experiences to the amount of coverage the planner can guarantee for that robot. In all of our experiments, it was obvious that as $\theta_{\max}$ increased, coverage decreased. Because we do not have a complete algorithm, it is not currently possible to address the question, even though it is true that for any $\theta_{\max}$, there exists some largest area we can cover. We offer some insight in determining those points too far from any edge to be covered, but do not consider the way in which navigation fails as error increases.

Bibliography

[1] E. U. Acar and H. Choset, *Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and voronoi diagrams*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001.

[2] ______, *Robust sensor-based coverage of unstructured environments*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001.

[3] Ercan U. Acar and Howie Choset, *Sensor-based coverage of unknown environments: Incremental construction of morse decompositions*, The International Journal of Robotics Research **21** (2002), no. 4, 345–366.

[4] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull, *Morse decompositions for coverage tasks*, The International Journal of Robotics Research **21** (2002), no. 4, 331–344.

[5] Ercan U. Acar, Howie Choset, Yangang Zhang, and Mark Schervish, *Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods*, The Int. Journal of Robotics Research **22** (2003), 441–466.

[6] *Autonomous Field Robotics Laboratory (AFRL)*, https://afrl.cse.sc.edu/afrl/home/ Last Accessed: 2018-10-28.

[7] Noa Agmon, Noam Hazon, Gal A Kaminka, and The MAVERICK group, *The giving tree: constructing trees for efficient offline and online multi-robot coverage*, Annals of Mathematics and AI **52(2-4)** (2008), 143–168.

[8] S. Akella, W. Huang, K. M. Lynch, and M. T. Mason, *Parts feeding on a conveyor with a one joint robot*, Algorthmica **26** (2000), no. 3, 313–344.

[9] S. Akella and M. Mason, *Posing polygonal objects in the plane by pushing*, International Journal of Robotics Research **17** (1998), no. 1, 70–88.

[10] James Arvo and David Kirk, *Fast ray tracing by ray classification*, SIGGRAPH, 1987.

[11] Prasad N Atkar, David C Conner, Aaron Greenfield, Howie Choset, and Alfred A Rizzi, *Uniform coverage of simple surfaces embedded in $\mathbb{R}^3$ for auto-body painting*, Proc. Workshop on the Algorithmic Foundations of Robotics, 2004.

[12] Gustavo S. C. Avellar, Guilherme A. S. Pereira, Luciano C. A. Pimenta, and Paulo Iscold, *Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time*, Sensors **15(11)** (2015), 27783.

[13] Richard Bellman, *Dynamic programming treatment of the travelling salesman problem*, J. ACM **9** (1962), no. 1, 61–63.

[14] R.-P. Berretty, K. Goldberg, M. Overmars, and F. Van der Stappen, *Trap design for vibratory part feeders*, International Journal of Robotics Research **20** (2001), no. 11, 0–0.

[15] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph theory 1736–1936*, Oxford(Clarendon Press), 1976.

[16] A. Blum, P. Raghavan, and B. Schieber, *Navigating in unfamiliar geometric terrain*, SIAM Journal on Computing **26** (1997), no. 1, 110–137.

[17] M. Blum and D. Kozen, *On the power of the compass (or, why mazes are easier to search than graphs)*, Proc. IEEE Symposium on Foundations of Computer Science, 1978, pp. 132–142.

[18] Prosenjit Bose and Marc Van Kreveld, *Generalizing monotonicity: on recognizing special classes of polygons and polyhedra*, International Journal of Computational Geometry & Applications **15** (2005), no. 06, 591–608.

[19] *Boston Dynamics Robots*, https://www.bostondynamics.com/robots Accessed: 2018-09-25.

[20] S. Brown and S. L. Waslander, *The constriction decomposition method for coverage path planning*, 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2016, pp. 3233–3238.

[21] A. Del Bue, M. Tamassia, F. Signorini, V. Murino, and A. Farinelli, *Visual coverage using autonomous mobile robots for search and rescue applications*,

2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Oct 2013, pp. 1–8.

[22] J. F. Canny, *The complexity of robot motion planning*, MIT Press, Cambridge, MA, 1988.

[23] B. Chazelle and L. G. Guibas, *Visibility and intersection problems in plane geometry*, Discrete and Computational Geometry **4** (1989), 551–589.

[24] Young-Ho Choi, Tae-Kyeong Lee, Sang-Hoon Baek, and Se-Young Oh, *Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009.

[25] H. Choset, E. Acar, A.A. Rizzi, and J. Luntz, *Exact cellular decompositions in terms of critical points of morse functions*, Proc. IEEE International Conference on Robotics and Automation, 2000, pp. 2270–2277.

[26] H. Choset and J. Burdick, *Sensor based motion planning: Incremental construction of the hierarchical generalized Voronoi graph*, International Journal of Robotics Research **19** (2000), no. 2, 126–148.

[27] ______, *Sensor based motion planning: The hierarchical generalized Voronoi graph*, International Journal of Robotics Research **19** (2000), no. 2, 96–125.

[28] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: Theory, algorithms, and implementations*, MIT Press, Boston, MA, 2005.

[29] Howie Choset, *Coverage for robotics - a survey of recent results*, Annals of Mathematics and Artificial Intelligence **31** (2001), 113–126.

[30] Howie Choset and Philippe Pignon, *Coverage path planning: The boustrophedon cellular decomposition*, International Conference on Field and Service Robotics (Canberra, Australia), 1997.

[31] X. Deng, T. Kameda, and C. H. Papadimitriou, *How to learn an unknown environment I: The rectilinear case*, Journal of the ACM **45** (1998), no. 2, 215–245.

[32] B. R. Donald, *On information invariants in robotics*, Artificial Intelligence **72** (1995), 217–304.

[33] L. E. Dubins, *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*, American Journal of Mathematics **79** (1957), no. 3, 497–516.

[34] Jack Edmonds and Ellis L. Johnson, *Matching, euler tours and the chinese postman*, Mathematical Programming **5** (1973), no. 1, 88–124.

[35] S. P. Engelson and D. V. McDermott, *Error correction in mobile robot map learning*, Proceedings 1992 IEEE International Conference on Robotics and Automation, May 1992, pp. 2555–2560 vol.3.

[36] M. Erdmann and M. T. Mason, *An exploration of sensorless manipulation*, IEEE Transactions on Robotics and Automation **4** (1988), no. 4, 369–379.

[37] M. A. Erdmann, *Using backprojections for fine motion planning with uncertainty*, International Journal of Robotics Research **5** (1986), no. 1, 19–45.

[38] ______, *Understanding action and sensing by designing action-based sensors*, International Journal of Robotics Research **14** (1995), no. 5, 483–509.

[39] L. Erickson, J. Knuth, J. M. O'Kane, and S. M. LaValle, *Probabilistic localization with a blind robot*, Proc. IEEE International Conference on Robotics and Automation, 2008.

[40] *FAA aerospace forecast*, https://www.faa.gov/data_research/aviation/aerospace_forecasts/ media/FY2016-36_FAA_Aerospace_Forecast.pdf, Accessed: 2018-09-25.

[41] *Farm Bot*, https://farm.bot/, Accessed: 2018-09-25.

[42] P. Fazli, A. Davoodi, P. Pasquier, and A.K. Mackworth, *Complete and robust cooperative robot area coverage with limited range*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010, pp. 5577–5582.

[43] Robert W. Floyd, *Algorithm 97: Shortest path*, Commun. ACM **5** (1962), no. 6, 345–348.

[44] P Mojiri Forooshani and M Jenkin, *Sensor coverage with a heterogeneous fleet of autonomous surface vessels*, IEEE International Conference on Information and Automation, 2015, pp. 571–576.

[45] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), Oct 1976, pp. 216–227.

[46] Y. Gabriely and E. Rimon, *Spiral-stc: an on-line coverage algorithm of grid environments by a mobile robot*, Proc. of the IEEE Int. Conf. on Robotics and Automation, vol. 1, 11-15 May 2002, pp. 954 – 960.

[47] Enric Galceran and Marc Carreras, *A survey on coverage path planning for robotics*, Robotics and Autonomous Systems **61** (2013), no. 12, 1258–1276.

[48] Michael R. Garey and David S. Johnson, *Computers and intractability; a guide to the theory of np-completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.

[49] A. S. Glassner, *Space subdivision for fast ray tracing*, IEEE Computer Graphics and Applications **4** (1984), no. 10, 15–24.

[50] K. Y. Goldberg, *Orienting polygonal parts without sensors*, Algorthmica **10** (1993), 201–225.

[51] Enrique Gonzalez, Oscar Alvarez, Yul Diaz, Carlos Parra, and Cesar Bustacara, *BSA: a complete coverage algorithm*, Proc. IEEE International Conference on Robotics and Automation, 2005.

[52] R. Graham, M.R. Garey, and D.S. Johnson, *Some np-complete geometric problems*, 8th Annual ACM Symposium on Theory of Computing, 1976, pp. 10–22.

[53] N. Hazon and G.A. Kaminka, *Redundancy, efficiency and robustness in multi-robot coverage*, IEEE International Conference on Robotics and Automation (ICRA), 2005, pp. 735–741.

[54] Michael Held and Richard M. Karp, *A dynamic programming approach to sequencing problems*, Proceedings of the 1961 16th ACM National Meeting (New York, NY, USA), ACM '61, ACM, 1961, pp. 71.201–71.204.

[55] W.H. Huang, *Optimal line-sweep-based decompositions for coverage algorithms*, Proc. the IEEE Int. Conf. on Robotics and Automation, vol. 1, 2001, pp. 27 – 32.

[56] *iRobot autonomous floor-cleaning robot*, https://patents.google.com/patent/US6883201B2/en, Accessed: 2018-10-13.

[57] *iRobot reports record fourth-quarter and full-year revenue*, http://media.irobot.com/2018-02-07-iRobot-Reports-Record-Fourth-Quarter-and-Full-Year-Revenue, Accessed: 2018-09-25.

[58] I. Kamon and E. Rivlin, *Sensory-based motion planning with global proofs*, IEEE Transactions on Robotics and Automation **13** (1997), no. 6, 814–822.

[59] I. Kamon, E. Rivlin, and E. Rimon, *Range-sensor based navigation in three dimensions*, Proc. IEEE International Conference on Robotics and Automation, 1999.

[60] Nare Karapetyan, Kelly Benson, Chris McKinney, Perouz Taslakian, and Ioannis Rekleitis, *Efficient multi-robot coverage of a known environment*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (Vancouver, BC, Canada), Sept. 2017.

[61] Nare Karapetyan, Jason Moulton, Jeremy Lewis, Alberto Quattrini Li, Jason O'Kane, and Ioannis Rekleitis, *Multi-robot Dubins Coverage with Autonomous Surface Vehicles*, IEEE International Conference on Robotics and Automation (Brisbane, Australia), May 2018, pp. 2373–2379.

[62] Peter Kimball, John Bailey, Sarah Das, Rocky Geyer, Trevor Harrison, Clay Kunz, Kevin Manganini, Ken Mankoff, Katie Samuelson, Thomas Sayre-McCord, Fiamma Straneo, Peter Traykovski, and Hanumant Singh, *The whoi jetyak: An autonomous surface vehicle for oceanographic research in shallow or dangerous waters*, Autonomous Underwater Vehicles (AUV), 2014.

[63] Chan Sze Kong, Ai Peng New, and Ioannis Rekleitis, *Distributed coverage with multi-robot system*, Proc. of the IEEE International Conference on Robotics and Automation, 2006, pp. 2423 – 2429.

[64] S. M. Kristek and D. A. Shell, *Orienting deformable polygonal parts without sensors*, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 2012, pp. 973–979.

[65] Ratnesh Kumar and Haomin Li, *On asymmetric tsp: Transformation to symmetric tsp and performance bound*, Numerische Mathematik (2002), 63–99.

[66] ______, *On asymmetric TSP: Transformation to symmetric tsp and performance bound*, Tech. report, University of Kentucky, Department of Electrical Engineering, Feb 2014.

[67] Jean-Claude Latombe, *Robot motion planning*, Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[68] A. Lazanas and J. C. Latombe, *Landmark-based robot navigation*, Proc. National Conference on Artificial Intelligence (AAAI), 1992.

[69] J. K. Lenstra and A. H. G. Rinnooy Kan, *Complexity of vehicle routing and scheduling problems*, Networks **11** (1981), no. 2, 221–227.

[70] J. S. Lewis, D. A. Feshbach, and J. M. O'Kane, *Guaranteed navigation with an unreliable blind robot*, Proc. IEEE International Conference on Robotics and Automation, 2010.

[71] Jeremy S. Lewis, William Edwards, Kelly Benson, Ioannis Rekleitis, and Jason M. O'Kane, *Semi-boustrophedon coverage with a dubins vehicle*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2017.

[72] Jeremy S. Lewis and Jason M. O'Kane, *Planning for provably reliable navigation using an unreliable, nearly sensorless robot*, International Journal of Robotics Research **32** (2013), no. 11, 1339–1354.

[73] LocalSolver, http://www.localsolver.com.

[74] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor, *Automatic synthesis of fine-motion strategies for robots*, International Journal of Robotics Research **3** (1984), no. 1, 3–24.

[75] V. J. Lumelsky and S. Tiwari, *An algorithm for maze searching with azimuth input*, Proc. IEEE International Conference on Robotics and Automation, 1994, pp. 111–116.

[76] M. Moll and M. Erdmann, *Manipulation of pose distributions*, International Journal of Robotics Research **21** (2002), no. 3, 277–292.

[77] Charles E. Noon and James C. Bean, *An efficient transformation of the generalized traveling salesman problem*, INFOR: Information Systems and Operational Research **31** (1993), no. 1, 39–44.

[78] Jerome Le Ny, Eric Feron, and Emilio Frazzoli, *On the dubins traveling salesman problem*, IEEE Trans. Automat. Contr. **57** (2012), 265–270.

[79] C. Ó. Dúnlaing and C. K. Yap, *A retraction method for planning the motion of a disc*, Journal of Algorithms **6** (1982), 104–111.

[80] J. M. O'Kane and S. M. LaValle, *Almost-sensorless localization*, Proc. IEEE International Conference on Robotics and Automation, 2005.

[81] ———, *Sloppy motors, flaky sensors, and virtual dirt: Comparing imperfect ill-informed robots*, Proc. IEEE International Conference on Robotics and Automation, 2007.

[82] José Manuel Palacios-Gasós, Zeynab Talebpour, Eduardo Montijano, Carlos Sagüés, and Alcherio Martinoli, *Optimal path planning and coverage control for multi-robot persistent coverage in environments with obstacles*, Proc. IEEE International Conference on Robotics and Automation, IEEE, 2017, pp. 1321–1327.

[83] Christos H. Papadimitriou, *The euclidean travelling salesman problem is np-complete*, Theoretical Computer Science **4** (1977), no. 3, 237 – 244.

[84] Trey Parker and Matt Stone, *Gnomes*, Dec 1998.

[85] Liam Paull, Carl Thibault, Amr Nagaty, Mae Seto, and Howard Li, *Sensor-driven area coverage for an autonomous fixed-wing unmanned aerial vehicle*, IEEE transactions on cybernetics **44** (2014), no. 9, 1605–1618.

[86] Preparata, P., Franco Shamos, and Michael Ian, *Computational geometry : An introduction*, Springer, 01 1985.

[87] Michael Yu Rachkov, Lino Marques, and AnÍbal T de Almeida, *Multisensor demining robot*, Autonomous robots **18** (2005), no. 3, 275–291.

[88] Ioannis Rekleitis, Jean-Luc Bedwani, Erick Dupuis, and Pierre Allard, *Path planning for planetary exploration*, Canadian Conference on Computer and Robot Vision (CRV) (Windsor, ON, Canada), May 2008, pp. 61–68.

[89] Ioannis Rekleitis, Vincent Lee-Shue, Ai Peng New, and Howie Choset, *Limited communication, multi-robot team based coverage*, Proc. of the 2004 IEEE Int. Conf. on Robotics and Automation, 2004.

[90] Ioannis Rekleitis, AiPeng New, Edward Samuel Rankin, and Howie Choset, *Efficient boustrophedon multi-robot coverage: an algorithmic approach*, Annals of Mathematics and AI **52(2-4)** (2008), 109–142 (English).

[91] A. Renzaglia, L. Doitsidis, S. A. Chatzichristofis, A. Martinelli, and E. B. Kosmatopoulos, *Distributed multi-robot coverage using micro aerial vehicles*, Mediterrean Conference on Control Automation (MED), June 2013, pp. 963–968.

[92] Scott D. Roth, *Ray casting for modeling solids*, Computer Graphics and Image Processing **18** (1982), 109–144.

[93] Nicholas Roy and Sebastian Thrun, *Coastal navigation with mobile robots*, Advances in Neural Processing Systems, 1999, pp. 1043–1049.

[94] Haydar Sahin and Levent Guvenc, *Household robotics: autonomous devices for vacuuming and lawn mowing*, IEEE Control Systems **27** (2007), no. 2, 20–96.

[95] K. Savla, F. Bullo, and E. Frazzoli, *The coverage problem for loitering dubins vehicles*, Decision and Control, 2007 46th IEEE Conference on, Dec 2007, pp. 1398–1403.

[96] Ketan Savla, Emilio Frazzoli, and Francesco Bullo, *On the point-to-point and traveling salesperson problems for dubins' vehicle.*, American Control Conference, June 2005, pp. 786–791.

[97] L Szirmay-Kalos and G Marton, *Worst-case versus average case complexity of ray-shooting*, Computing **61(2)** (1998), no. 2, 103–131.

[98] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler, *Sensor planning for a symbiotic UAV and UGV system for precision agriculture*, IEEE Transactions on Robotics **32** (2016), no. 6, 1498–1511.

[99] B. Tovar, L. Guilamo, and S. M. LaValle, *Gap Navigation Trees: Minimal representation for visibility-based tasks*, Proc. Workshop on the Algorithmic Foundations of Robotics, 2004.

[100] Ben Tribelhorn and Zachary Dodds, *Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education*, Proc. IEEE International Conference on Robotics and Automation, 2007.

[101] A. F. van der Stappen, R.-P. Berretty, K. Goldberg, and M. H. Overmars, *Geometry and part feeding*, Sensor Based Intelligent Robots, 2000, pp. 259–281.

[102] J.R. VanderHeide, *Terrain coverage of an unknown room by an autonomous mobile robot*, Tech. report, Michigan State Univ, 12 1995.

[103] K. Sun V.J. Lumelsky, S. Mukhopadhyay, *Dynamic path planning in sensor-based terrain acquisition*, 1990, pp. 462–472.

[104] D. E. Whitney, *Real robots don't need jigs*, Proc. IEEE International Conference on Robotics and Automation, 1986.

[105] Gerhard J. Woeginger, *Combinatorial optimization - eureka, you shrink!*, Combinatorial Optimization - Eureka, You Shrink! (Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, eds.), Springer-Verlag New York, Inc., New York, NY, USA, 2003, pp. 185–207.

[106] Thaddeus A. Roppel Xin Yu and John Y. Hung, *An optimization approach for planning robotic field coverage*, IECONN, 2015.

[107] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis, *Efficient complete coverage of a known arbitrary environment with applications to aerial operations*, Autonomous Robots **36** (2014), no. 4, 365–381 (English).

[108] Z. Yao, *Finding efficient robot path for the complete coverage of a known space*, Proc. of the IEEE Int. Conf. on Robotics and Automation (Beijing, China), 2006, pp. 3369–3374.

[109] Xiaoming Zheng, Sonal Jain, Sven Koenig, and David Kempe, *Multi-robot forest coverage*, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.