Doctoral Thesis

Constraint-Based Coverage Path Planning: A Novel Approach to Achieving Energy-Efficient Coverage

Christopher Krzysztof Tomaszewski
The Robotics Institute
Carnegie Mellon University

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

December 2020

Tech Report Number: CMU-RI-TR-20-60

Abstract

Despite substantial technological progress that has driven the proliferation of robots across various industries and aspects of our lives, the lack of a decisive breakthrough in electrical energy storage capabilities has restrained this trend, particularly with respect to mobile robots designed for use in unstructured and unknown field environments. The fact that these domains are often less accessible and previously unexplored makes them a perfect candidate for the use of robots, but at the same time raises the stakes of failures such as premature energy depletion, which can result in various irrecoverable scenarios leading to loss of time and equipment. If we are to successfully leverage robots for these promising applications in the absence of a revolutionary development in battery technology, it is clear we must consider alternative means of addressing the energy storage issue.

Fortunately, many of the features that contribute to the difficulty of real-world field scenarios can actually be a source of ambient energy that can be captured and exploited to extend operation beyond what is possible using limited on-board energy supply. In fact, humans have been harnessing the power of wind and currents to travel the oceans for centuries and continue to take into account ambient energy when performing various activities, either by instinct or design. The premise of this thesis is that this same thinking can be extended to planning for mobile robots in the field in order to mitigate the shortcomings of present day energy storage technology and open up further opportunities for robotic exploitation.

In particular, this thesis considers the problem of energy-efficient Coverage Path Planning (CPP) in river domains where ambient energy is present in forms such as wind, currents, or elevation change. Instead of formulating a specialized monolithic algorithm to achieve energy-efficiency in a given set of scenarios, we instead develop a novel constraint-based coverage path planning (CB-CPP) framework, which breaks down the planning process into discrete stages that can be optimized and tuned individually. This approach allows us to specifically consider the energy-use implications of the order in which different parts of the region are covered as well as the vehicle's direction of travel in each region. Finally, we propose several new energy-efficient coverage planners implemented within our CB-CPP framework that leverage these insights. We demonstrate and validate this work on a variety of simulation scenarios across domains and with real-world experiments with an autonomous surface vehicle deployed in rivers.

Acknowledgements

When I first started at Carnegie Mellon as an undergrad in 2008, I had no idea I would end up sitting in front of my computer writing the acknowledgements for my PhD thesis over 12 years later, at the height of a global pandemic no less. During my extended tenure at CMU, I have learned a tremendous amount, forgotten much of it, and built a wealth of experience that has profoundly impacted my life and will undoubtedly continue to do so for years to come. Along the way, I have been fortunate to interact with and receive support from some of the best people on Earth, without whom I would never have arrived at this point. Although it is impractical to thank everyone who has aided me on this journey, in the spirit of taking on impossible tasks, I would like to make an attempt.

First and foremost, I would like to thank my daughter Zoe. Although you were born a few weeks after my defense, your impending arrival has been the strongest motivation for me to buckle down and finish my thesis. I look forward to shepherding you through your own life journey and watching you learn and discover all the incredible things in this world.

Next, it is imperative that I thank my magnificent wife Tània, who has been a constant source of support and encouragement for years and patiently put up with me playing with robots in rivers while she worked a real job and carried our baby. I love you.

I would also be remiss if I did not thank all of my family, who tirelessly supported, encouraged, and pushed me throughout this process. In particular, I am forever grateful to my parents, who selflessly assisted me at every turn and were instrumental to all my success. A special thanks to my sister, uncle, grandparents, and parents-in-law as well.

On the academic side of things, I want to offer my sincerest thanks to all the members of my committee whose time and efforts were critical to the completion of this work. Paul, for building that first airboat and then bringing me on to make it float, thereby kicking off a decade-long investigation into the various failure-modes of autonomous surface vehicles, which has shaped me into the roboticist I am today. John, for his unwavering support, advice, and guidance in transforming my collection of misadventures in the field into a coherent piece of research. George, for introducing me to controls as an undergrad and continuously asking the tough questions that made me reexamine my work and strive to improve it. Mel, for inspiring my interest into measuring river currents as an undergrad and his insightful suggestions that influenced many of my ideas. And finally, Jordi, for giving me the chance to share my work and reminding me to consider the applications of my research.

Of course it's hard to go for over a decade and not make any friends, and I've made some great ones. Thank you Aaron, Dev, Grant, Gunnar, Jason, Kendra, Nate, Pras, Sean, Shantanu, Shaurya, Shawn, and Steve. Whether it was putting up with me as a roommate, offering some sage advice (technical or life), helping with field tests, or just being there to hang out, you all contributed to getting me across the finish line and I am truly grateful. Thank you also to all my colleagues within the Robotics Institute and CMU as well as the incredible staff, particularly Chuck, Rachel, and Suzanne.

Finally, thanks to my bulldog Winston, who faithfully kept me company and whose rhythmic snoring provided the perfect backdrop for writing my thesis into the early hours of the morning.

Contents

Chapter 1

Introduction

Continued technological advances coupled with breakthroughs in manufacturing have sped the adoption of autonomous systems to the point that new cars are expected to drive themselves and everyday encounters with cleaning robots are nothing to write home about. One area that does not seem to have kept pace with this progress, however, is energy storage, which remains a considerable obstacle for all mobile robot applications. Although robots designed for situations where the implications of premature energy depletion are trivial or easily addressed (e.g., home robotic vacuum cleaners) have been able to gain popularity despite this issue, energy constraints have severely impacted the usage of robots in situations where power failure can have more dire consequences (e.g., a UAV running out of power mid-flight). Fortunately, in many domains of interest ambient energy is present in various forms such as wind, waves, currents, and elevation change, which when properly exploited can help surmount these energy storage limitations and enable broader use of robots beyond the realm of cleaning carpets.

One clear robotic application that would benefit from such improvements is the gathering of environmental data. Manual collection of environmental data over a large area can be a time-consuming, costly, and even dangerous process, making it a perfect candidate for automation with mobile robots. Unfortunately, this type of data often needs to be collected at a dense set of points throughout the survey region to adequately capture certain features that may go unnoticed if the area is explored more selectively. This complete coverage requirement unavoidably translates to an extended deployment, which can be problematic for mobile robots with a limited on-board energy supply. Although much research into the Coverage Path Planning (CPP) problem has been done in pursuit of time- and length-optimal planning algorithms, there has not been a commensurate effort to develop CPP techniques that minimize the energy requirements of the resulting coverage plan. The result is that while there exist specialized CPP algorithms that claim efficiency according to various metrics for specific applications or scenarios, they are often difficult to generalize to other applications and may not be easily extensible to leverage additional domain knowledge or insights to further improve performance.

1.1 Problem Statement/Thesis

This thesis seeks to address this gap by first formulating a new framework that breaks up the CPP process into a set of discrete stages that can be optimized individually and then developing a novel energy-efficient CPP algorithm using this architecture. In particular, we investigate how exploiting overlap in sensor coverage and adjusting the direction and sequence of different segments of the coverage path can result in improved energy-efficiency in the presence of fluid flow fields such as wind or currents. More formally, the thesis of this work is:

> A coverage path planner can be constructed in such a fashion as to take advantage of ambient domain energy by strategically selecting the order and direction path segments are visited in, as well as opportunistically exploiting sensor coverage overlap to plan movement through easier regions or along shorter routes.

In order to demonstrate the generality of our approach, we include simulation results from different domains and vehicles across a variety of scenarios. For practicality reasons, however, we restrict our real-world experiments to a single Autonomous Surface Vehicle (ASV) platform deployed in a set of straightforward scenarios in a river.

1.2 Contributions

The key contributions of this thesis are:

1. CB-CPP, a Constraint-Based Coverage Path Planning framework, which produces planning algorithms that assemble coverage paths from extensible path constraints and can reason about the constraints individually during the planning process in order to achieve highly specific performance objectives.
2. A specialized path constraint layout module within the CB-CPP framework for use in river domains that leverages overlap in the coverage envelope while distributing path constraints throughout the target area to shorten the overall path and optimize impacts of surface currents on the vehicle.
3. A pair of path constraint refinement and sequencing modules that can be applied to different constraint layouts and function together to reduce energy usage of coverage paths across regions with moving fluids by intelligently guiding vehicle movement according to the flow along each path constraint.
4. A set of new CPP algorithms utilizing combinations of the above modules built on top of a boustrophedon planner implemented within the CB-CPP framework.
5. A real-world implementation of the above on an autonomous surface vehicle.

1.3 Thesis Outline

The remainder of this section will provide a brief outline of the thesis. Chapter 2 begins with a broad introduction to the Coverage Path Planning problem, with a detailed survey of conventional methods, and concludes with a discussion of state of the art with regards to energy-efficient coverage planning. Chapter 3 introduces the Path Constraint abstraction and presents CB-CPP, our proposed coverage path planning framework. Chapter 4 details the specifics of two coverage planning optimizations developed within the CB-CPP framework with the objective of enabling energy-efficient coverage of river domains in the presence of currents. Chapter 5 describes the simulation platform we built to evaluate the efficacy of our proposed optimizations, and presents results from several simulation scenarios that support our hypothesis. Chapter 6 details the implementation of the purpose-built platform used to field test our algorithms, and provides an analysis of results from field experiments and simulation. Finally, Chapter 7 concludes with closing remarks and a discussion of avenues for future investigation.

Chapter 2

Background

The primary contribution of this thesis is the design of a novel coverage path planning framework that enables the adaptation of energy-efficient techniques from the realm of point-to-point planning to the coverage planning domain. This chapter sets the stage for our discussion of this work, and motivates the development of our framework through an examination of existing approaches to coverage path planning, energy-efficient path planning, and energy-efficient coverage path planning. Through this literature survey, we aim to expose a clear gap in the state of the art where established methods for achieving energy-efficiency during path planning have not been effectively applied to planning for coverage.

During our discussion of existing energy-efficient coverage planning techniques, we identify a dearth of methods that optimize for energy usage domains involving moving fluids, such as winds or currents. These domains are of particular interest as far as energy conserving algorithms are concerned because they offer abundant ambient energy in the environment that can assist or hinder motion. The lack of research interest in this specific problem motivates the second contribution of this thesis: the development of an energy-efficient approach to coverage of riverine domains, which exploits variations in surface currents across the survey region as well as typical geometric features of rivers in order to reduce energy consumption. The algorithm proposed is implemented within the new coverage planning framework introduced in this thesis.

The remainder of this chapter is organized into three primary sections covering the seminal techniques and works dealing with coverage path planning, energy-efficient path planning, and a combination of the two, energy-efficient coverage path planning. In the first section we introduce the coverage path planning problem and present a spectrum of different methods that have been developed over the years. In the next section, we examine the task of optimizing paths for energy consumption through a variety of approaches at different levels of the planning hierarchy. Finally, we discuss the set of existing work involving energy-efficient coverage planning for the limited set of domains and scenarios where it has been considered.

2.1 Coverage Path Planning

Although specific formulations vary across implementations, fundamentally, the problem of Coverage Path Planning (CPP) is to determine an appropriate coverage path given a robot specification, the reach of its sensor or actuator payload, and the target region. The desirable criteria for a coverage path solution were defined by Cao, Huang, and Hall in some of the earliest work on CPP and continue to influence algorithms developed today [11]. The requirements they laid out are as follows:

1. The robot must move through an entire area (i.e., cover the whole region).
2. The robot must fill the region without overlapping paths.
3. Continuous and sequential operation without any repetition of paths is required.
4. The robot must avoid all obstacles.
5. Simple motion trajectories (e.g., straight lines or circles) should be used for simplicity in control.
6. An ”optimal” path is desired under the available conditions.

Clearly it is seldom possible to completely satisfy all these criteria and therefore it is typically necessary to prioritize a subset over the others. In fact, unless the target coverage region can be exactly covered by a continuous sweep of the robot’s payload footprint, some overlapping must occur to fully cover the region or certain areas will be missed. Many of the developed CPP algorithms balance the tradeoffs between these criteria to achieve the desired performance for specific applications or scenarios, resulting in a diverse set of specialized techniques and algorithms. A broad set of these approaches have been summarized and discussed in the surveys of Choset [15], Galceran and Carreras [30], and most recently Bormann et al. [9].

Although its criteria are often defined in the context of guiding a mobile robot, CPP is not strictly a robotics problem and is related to several research areas across computer science and manufacturing. Several clear parallels can be drawn between CPP and the famous Travelling Salesman Problem (TSP), which is commonly stated as follows: given the distances between each pair of n cities, find a tour (i.e., a simple path visiting all cities) that minimizes the total distance travelled. Of particular interest is the Euclidean case of TSP, where cities are points on a map and the distance between them is defined by the usual Euclidean metric, as well as its generalization, the Covering Salesman Problem (CSP), where the determined tour must only visit a neighborhood defined around each city [7]. If the polygon defining our robot’s reach is chosen to represent each city’s neighborhood and the cities are distributed densely throughout the target region as well as along its boundary, the CPP problem can be transformed to CSP. Unfortunately Euclidean TSP and consequently CSP, are NP-Complete [74], which suggests finding an optimal solution using this approach will require dramatically increasing amounts of computation as the dimension of the problem increases. In fact, in some of their later work, Arkin et al. show that the ”lawn mowing” or ”milling” problem of finding a shortest path to cut all the grass or material from a given region is itself NP-Hard [6]. As robots take over tasks such as mowing lawns and vacuuming floors, these problems have shifted to become classic applications for CPP.

As alluded to above, the CPP problem is also closely related to the problem of automated tool path generation for manufacturing and fabrication. There is a significant amount of work that has come out of the CAD community regarding the milling or pocket machining problem, which is informally stated as follows: given the boundary contours enclosing a multiply-connected planar area and the size of a tool, determine the tool path for machining the pocket [35]. Clearly if we define the size of the tool as the reach of the robot, the CPP problem can be transformed to the pocket machining problem, although solutions to the latter often involve additional considerations specific to machining, such as spindle direction and surface quality. The recent rise in popularity of 3D printing has also been accompanied by a growing interest in determining the optimal tool path fill pattern from within the additive manufacturing community [32]. This problem is essentially the opposite of the pocket machining problem, where material is being added instead of removed in order to build up a 3D object layer by layer. Beyond the clear parallels to the CPP problem, it is interesting to note that the deleterious impact of several tool path characteristics such as sharp turns or corners, which can lead to discretization artifacts and degrade fill quality [44], can have similarly negative consequences on a coverage path where the robot must change speed and direction rapidly. As increasingly sophisticated fill pattern tool path algorithms are developed [119], they continue to influence the development of novel CPP approaches [114].

Before diving more deeply into an examination of CPP methods, it is useful to first discuss a few characteristics commonly used to distinguish these algorithms. Firstly, CPP algorithms can generally be classified as either heuristic or complete, depending on whether they can provably guarantee complete coverage of the free space. Additionally, CPP methods can be partitioned into either offline or online approaches, where offline algorithms rely on full a priori knowledge of the environment while online methods do not assume full prior domain knowledge and instead rely on real-time sensor measurements. This classification was originally proposed by Choset in his survey of CPP methods and has largely endured to this day [15].

In his discussion of complete CPP algorithms, Choset points out that in order to achieve some sort of provable guarantee of coverage, many of these techniques rely on a cellular decomposition of the target region into cells such that coverage of each cell is ”simple.” Provably complete coverage can then be guaranteed by planning a path that ensures the robot visits each cell in the decomposition. Choset then continues to introduce three classes of cellular decomposition: approximate, semi-approximate, and exact. According to his definition, an approximate cellular decomposition is a fine-grid representation with uniform cells whose union only approximates the target region, while an exact cellular decomposition is a set of non-intersecting regions whose union fills the target environment. The semi-approximate decomposition is somewhat of a middle ground to host methods relying on partial discretization, such as the work of Hert and Lumelsky, which decomposes the target region into cells of fixed width but arbitrary top and bottom boundaries [36].

Although adequate at the time, Choset’s CPP algorithm classification scheme falls short when it comes to including complete coverage methods that do not rely on decomposing the target region into cells, such as those inspired by space-filling curves for tool path generation. In order to remedy this and streamline our further discussion of CPP methods, we propose a modification to Choset’s original methodology; while each CPP algorithm will still be classified as heuristic or complete as well as offline or online, it will additionally be placed along a decomposition spectrum that indicates the level of decomposition performed on the target region by the algorithm. Using this updated classification, we can place grid-based methods that discretize the target area into uniform robot-sized cells on one end of the spectrum and global methods that don't partition the target region on the other. An illustration of our proposed decomposition spectrum for coverage path planning techniques is provided in figure 2.1. For our following discussion of CPP methods, we will discuss algorithms spanning three approximate regions of the spectrum: Grid-based Methods, Cellular Decomposition Methods, and Global Methods.

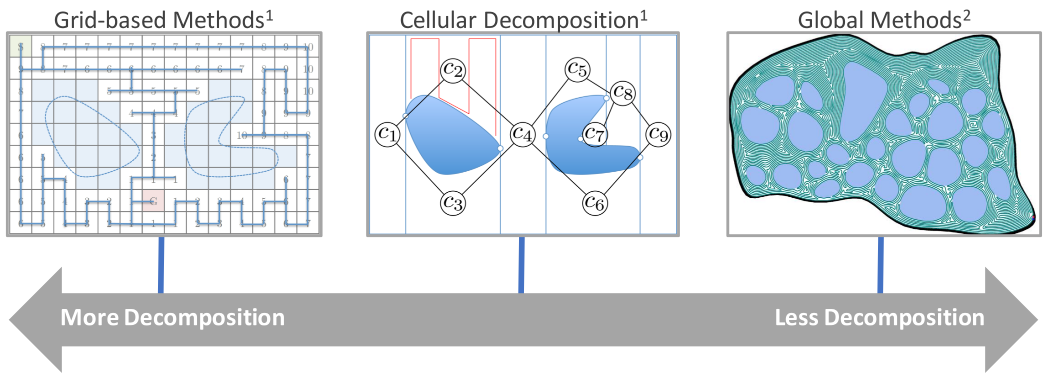


Figure 2.1: Decomposition Spectrum of Coverage Path Planning Techniques. Images adapted from (1) Galceran and Carreras' survey of CPP methods [30], and (2) Zhao et al.'s paper on Fermat spirals as tool paths for fabrication [119]

2.1.1 Grid-Based Methods

We begin our overview of CPP methods at the most granular end of the decomposition spectrum with a discussion of grid-based methods. Algorithms at this end of the spectrum decompose the target area into uniform grid cells, which are typically the size of the robot's footprint or reach so that when the robot enters a cell it is considered covered. After applying this decomposition, the CPP problem reduces to determining a path that visits each cell.

The grid representation was first proposed by Moravec and Elfes to map an indoor environment using a sonar ring mounted to a robot [26,70]. In their representation, each grid cell has an associated value between -1 and 1 where negative and positive values indicate a cell is likely empty or occupied respectively. The magnitude of each cell's value represents the certainty of the information, and a 0 value signifies the state of that cell is unknown.

The shape of the grid cells used for grid decomposition does not necessarily need to be rectangular. In their work Oh et al. propose a triangular decomposition, which has the advantage of providing a higher resolution and higher connectivity than grids built on rectangular cells of similar size [71]. While a rectangular cell within a grid is typically connected to 4 or 8 of its neighboring cells, each triangular cell in the proposed decomposition is connected to 12 adjacent cells, which can lead to smoother paths and increased flexibility during planning but can also increase computational requirements. Another example of a non-rectangular grid cell is the exact hexagonal decomposition proposed by Paull et al. in their work on information gain-based coverage planning for autonomous underwater vehicles (AUVs) outfitted with side scan sonars [75,76]. An interesting result of using hexagons is that, unlike 8-connected square grids or 12-connected triangular grids, the distance between the centers of any two neighboring hexagonal cells is the same, which can simplify the objective function used during planning. Additionally, by sizing the hexagons appropriately, their application involving side-looking sensors allows for coverage of two additional cells whenever the AUV moves from one cell to another.

Although the choices of grid shape, resolution, and connectivity can have serious implications, decomposing the target region into a suitable grid is just a preliminary; the substance of grid-based CPP methods lies in connecting the grid cells into a suitable path that achieves the coverage objectives. The earliest published work formally proposing a grid-based coverage planning algorithm was that of Zelinksy et al., in which the authors present a technique based on propagating a wave front throughout the grid map to assign values to each free-space cell and then using these values to construct a coverage path [117]. The wave front begins by visiting a specified goal cell and repeatedly expands to the cells adjacent to the current set of visited cells, while assigning values based on some metric. When this metric is the length of the shortest path from the current cell to the goal cell, this wave front propagation is also known as the distance transform, and can be combined with a gradient descent approach to plan point-to-point paths [39]. In his doctoral work, Zelinsky also introduced the ”path transform,” a modification to the distance transform that includes a factor representing the discomfort exerted by nearby obstacles [118]. In their proposed grid-based coverage planner, Zelinsky et al. apply either the distance or path transform, then perform a pseudo gradient ascent from a specified starting cell by continuously moving towards the highest-value neighboring cell that has not yet been visited [117]. Though either transform will work, the authors note that the path transform produces better coverage paths that tend to follow the contours of the target region.

One inconvenient aspect of Zelinsky et al.'s approach is that it is a strictly offline method that requires full a priori knowledge of the target region, which is often unavailable in real-world situations. To address this shortcoming, numerous online grid-based methods have been developed that can collect information about obstacles during the coverage process. In their work Shivashankar et al. present several strategies for online coverage in unknown environments where a robot builds a mental map of the environment as it chooses directions to move [89]. One of the strategies they develop is a generalization of Zelinsky et al.'s wave front algorithm, where the technique is repeatedly applied with the robot's mental map to plan paths to yet uncovered cells.

Another interesting online grid-based CPP approach is the Spiral Spanning Tree Covering (Spiral-STC) algorithm developed by Gabriely and Rimon [27]. This technique uses a square grid decomposition with a $2D$ cell size, where D is the size of the robot or tool. Each cell is further divided into four square subcells of size D than can be considered covered as soon as the robot visits them. To build the coverage path, this algorithm incrementally constructs a spanning tree of the $2D$ cells within the grid while moving along the subcells to one side of the spanning tree. The result is that the robot circumnavigates the spanning tree constructed on the $2D$ cell grid. One disadvantage of this method is that partially occupied $2D$ cells are discarded and not covered by the planner, which can lead to poor coverage performance when the target area contains features smaller than $2D$. Gabriely and Rimon address this shortcoming in their followup work, where they cover partially occupied $2D$ cells by modifying the coverage path of the robot through those cells at the cost of some coverage overlap [28].

In fact, the coverage of partially obstructed cells is a common challenge many grid-based CPP methods must contend with. In their initial work, Gonzalez et al. devise the Backtracking Spiral Algorithm (BSA) for online CPP, based on the idea of filling simple regions using a spiral pattern and then linking these regions using a backtracking mechanism [33]. Although their approach uses a higher-resolution grid decomposition than Spiral-STC with a cell size equal to that of the robot, it fails to handle partially occupied cells in its original incarnation. Gonzalez et al. address this issue in their later work where they extend BSA to perform a wall-following behavior whenever an obstacle is encountered while executing the typical spiral fill pattern [34]. The authors present experimental data to show that this extension can improve the coverage rate of BSA by up to 30%.

Building on the techniques designed for Spiral-STC and Extended BSA, Choi et al. develop an online complete coverage planning algorithm based on Linking Spiral Paths using a Constrained Inverse Distance Transform (LSP-CIDT) [13]. Like Extended BSA, LSP-CIDT implements a similar spiral method of filling simple regions and executes a wall-following behavior whenever obstacles are encountered in order to cover partially obstructed cells. Instead of the memory-intensive backtracking mechanism proposed for BSA, however, LSP-CIDT introduces a new Constraint Inverse Distance Transform (CIDT), which is used to find the shortest path from the current cell to the entry cell of the closest uncovered region. In addition, LSP-CIDT proposes aligning the map's coordinate system along the first wall detected, which can reduce excessive turns due to the discretization inherent to grid decomposition. This trick is particularly effective when covering typical rectangular rooms but cannot completely eliminate the excessive turn effect for other polygonal areas.

Lee et al. further extend this work in the Linked Smooth Spiral Path (LSSP) algorithm [54]. This algorithm uses a higher-resolution grid decomposition where each cell is smaller than the robot and a new technique for generating smoother spiral paths. Individual spiral patterns are linked together using the CIDT approach developed for LSP-CIDT. The improved resolution and spiral path generation allow LSSP to generate significantly smoother paths than LSP-CIDT and essentially eliminate the excessive turns due to grid discretization for any shape of target region.

Given the dense interconnected nature of the cells within a typical grid decomposition, it should come as no surprise that a number of grid-based CPP methods adopt a neural network approach. In their work, Yang and Luo propose an online neural network CPP approach built on a grid decomposition with square robot-sized cells [60,115]. Each grid cell is associated with a neuron that is connected to its eight neighbors and exhibits dynamics governed by a shunting model based on the membrane equation originally developed by Hodgkin and Huxley [37]. This model is designed such that the output value of all neurons at a given instant (i.e., the activity landscape) attracts the robot to uncleaned areas while repelling the robot from cleaned areas and obstacles. The robot is assumed to be outfitted with sensors that can detect obstacles and whether cells are clean or unclean up to some fixed range away, but is not provided with any prior knowledge of the environment. This sensor information is used to dynamically build a map of the target region, which together with the current state of the robot, is used to compute the neural activation and guide coverage by the vehicle. An advantage of this method over those previously discussed is that is is robust to changes in the environment during operation (e.g., dynamic obstacles).

Although the neural network approach to grid-based methods has several advantages, the computational burden of these methods can necessitate the use of more powerful and expensive hardware. In their work, Qiu et al. build on the biologically inspired neural network approach discussed above by adding a local path planner that determines the robot's next move [79]. In their approach the neural network is still used to model the environment but rather than compute the next position using the neural activation, a heuristic planner generates a local optimal plan. In addition to reducing the computational load, this technique is shown to produce shorter coverage paths with fewer turns and less coverage overlap than the original neural network approach when moving obstacles are encountered during coverage.

Due to its intuitive nature and ease of implementation, the grid decomposition continues to be a popular tool in CPP for many applications. It does, however, suffer from several shortcomings that render it an inefficient choice in certain circumstances. First of all, a typical grid map suffers from exponential growth of memory usage as the target area gets larger because the size of the cells remains uniform throughout the decomposition, regardless of the environment's complexity [94]. Furthermore, grid-based methods typically require accurate localization in order to maintain the map's coherency [12,95]. Finally, since finding a path to visit all cells within a grid decomposition is reducible to the path version of Euclidean TSP, a known NP-complete problem, any optimal solution is likely to require significant computational resources that scale sharply as the size of the problem (i.e., the number of grid cells) increases. Fortunately, these drawbacks suggest a possible solution; by increasing the size of the cells used in the decomposition, removing the constraint that cells be uniform, and designing a simple procedure to cover each of these cells as they are visited, both the memory usage and total cell count can be substantially reduced. This idea forms the crux of the methods discussed in the next subsection, which covers CPP algorithms that occupy the middle ground of the decomposition spectrum.

2.1.2 Cell Decomposition Methods

Moving along the decomposition spectrum of CPP algorithms we shift our attention to less granular techniques, sometimes known as exact cellular decomposition but which we will henceforth refer to as cell decomposition methods. This class of planners relies on partitioning the free space of the target coverage area into a set of simple non-intersecting regions called cells, where each cell is "easy" to cover using simple motions such as a back-and-forth sweep or spiral pattern. A pair of cells within this decomposition is said to be adjacent if they share a common boundary. This fact can be used to build an adjacency graph representation of the target environment where each cell is a graph node and each pair of adjacent cells is connected with an edge in the graph. Given this representation, the CPP problem is reduced to planning an exhaustive walk through the adjacency graph that visits each node and uses the defined simple motions to cover the corresponding cell. Although this problem is still related to TSP, the larger cells used in this decomposition greatly reduce the number of cells needed to represent large target regions when compared to grid-based methods, thereby making the search for an optimal path more computationally tractable.

Due to the well studied nature of TSP, much of the research into cell decomposition methods for CPP tends to focus on the process of decomposing the target region into suitable cells. Perhaps the simplest such approach is known as the trapezoidal decomposition [51] or slab method [77], where a line, deemed a slice, is swept through the target region populated with polygonal obstacles. Whenever a slice intersects the vertex of an obstacle, additional cells are defined, ultimately resulting in a decomposition where each cell is a trapezoid. Trapezoidal cells are very convenient for coverage as a simple back-and-forth path, also known as a boustrophedon path after the pattern an ox follows while plowing a field, can be computed to guide a robot to every location within the cell. Although the original approach was an offline algorithm, VanderHeide and Rao extended the trapezoidal decomposition in their work to implement online coverage of initially unknown environments [104].

One drawback of the trapezoidal decomposition is that it tends to produce many cells due to the fact that it exclusively creates convex cells. This is undesirable because generally the more cells are present in the decomposition, the longer the final coverage path will be. Furthermore, many nonconvex cells can also be completely covered by simple motions, which suggests the cells from a trapezoidal decomposition could somehow be combined to improve the final coverage path. This is essentially the approach developed by Oksanen and Visala in their work on CPP for agricultural applications [72]; after applying a standard trapezoidal decomposition on the target region, their algorithm tries to merge groups of adjacent trapezoidal cells into larger cells according to a couple of simple rules and a tuned parameter indicating how rectangular each cell is.

Rather than attempting to recombine suitable cells following an initial decomposition, another common technique to reduce the final number of cells is to modify the decomposition process to create fewer cells directly. Perhaps the best-known method that espouses this approach is the boustrophedon cellular decomposition, originally developed by Choset and Pignon [14, 16]. As its name suggests, this method decomposes a target region into cells particularly suited to coverage with a boustrophedon pattern. In order to accomplish this, a slice is again swept through the target region; however, this time additional cells are only created if the slice intersects an obstacle at a single point, also known as a critical point. This approach generally results in fewer cells than a trapezoidal decomposition, which can reduce the length of the final coverage path. Just like the trapezoidal method, the boustrophedon decomposition assumes obstacles are polygonal and that the terrain is known a priori.

In their later work, Acar et al. develop a new decomposition approach in an effort to generalize the boustrophedon method to smooth and polygonal workspaces [3]. Because it is based on finding the critical points of Morse functions [68], this new technique and its variants became known as Morse decompositions. Although the boustrophedon decomposition is a special case of a Morse decomposition, the latter has several advantages and additional capabilities. Not only can Morse decomposition handle non-polygonal obstacles, but the specific Morse function used can influence the shapes of the resulting cells in the decomposition (e.g., circles, squares, triangular slices, etc). Furthermore, Acar and Choset were able to devise a method to perform coverage of planar spaces by detecting critical points using sensory range information and an algorithm using motion templates that ensures all these points are discovered within the target area, thereby allowing for complete coverage online [1, 2].

Although it is capable of being adapted to a variety of scenarios and applications, one area in which the Morse decomposition method falls short is rectilinear environments, where critical points cannot be determined. An alternative approach based on the detection of natural landmarks was developed by Wong and McDonald to function on a larger variety of environments, including those with polygonal, elliptical, and rectilinear obstacles [111–113]. Like the boustrophedon decomposition, the landmark method performs its decomposition by sweeping a slice through the target area, albeit with different events used to determine cell boundaries. Because these events or ”landmarks” can be detected using a combination of range measurements, sensor readings, and odometry, both offline and online versions of this algorithm are possible.

For many of the methods discussed, a slice of some sort is swept through the target region during decomposition. By changing the direction of this sweep, the resulting cell decomposition for a given area can vary substantially. It is not surprising then that considerable research has gone into investigating how sweep direction can be optimized to improve the final coverage path. In his work, Huang develops the minimum sum of altitudes (MSA) decomposition, which decomposes the region multiple times with sweep directions perpendicular to each edge in the area, uses dynamic programming to combine these cells into subregions, and then assigns a sweep direction to each subregion [38]. Revisiting the work of Oksanen and Visala presents another example of optimizing the cell decomposition by adjusting the sweep direction [72]. In order to fully decompose the target area, their trapezoidal split and merge technique is repeatedly applied along different sweep directions until the cell with the best score is found. This cell is then removed from the target coverage area and the process is repeated until the entire area has been decomposed into cells.

As alluded to above, the CPP problem is also closely related to the problem of automated tool path generation for manufacturing and fabrication. There is a significant amount of work that has come out of the CAD community regarding the milling or pocket machining problem, which is informally stated as follows: given the boundary contours enclosing a multiply-connected planar area and the size of a tool, determine the tool path for machining the pocket [35]. Clearly if we define the size of the tool as the reach of the robot, the CPP problem can be transformed to the pocket machining problem, although solutions to the latter often involve additional considerations specific to machining such as spindle direction and surface quality. The recent rise in popularity of 3D printing has also been accompanied by a growing interest in determining the optimal tool path fill pattern from within the additive manufacturing community [32]. This problem is essentially the opposite of the pocket machining problem, where material is being added instead of removed in order to build up a 3D object layer by layer.

As mentioned in our introduction to this section, many techniques for automated tool path generation are directly applicable to the CPP problem. Though many of these methods tend to avoid explicit decomposition of the target region, there are several approaches that could be considered cell decomposition methods. One example of such an algorithm is the tool path generation strategy for wire and arc additive manufacturing that was developed by Ding et al. [22]. Critical to their approach is a "divide-and-conquer" strategy, which essentially decomposes the target region into convex polygons using a technique based on removing "notches," i.e., vertices with an internal angle greater than π. After the decomposition, a coverage path for each cell is generated using combination of contour and zigzag pattern strategies. These individual cell coverage paths are finally joined into a single closed curve that fully covers the target region.

Although many cellular decomposition techniques used for CPP take a mainly geometric approach that strives to reduce the total number of cells or generate cells that can be more efficiently covered, there are other strategies from related research areas that could be used in a cell decomposition CPP algorithm. For example, methods of map segmentation into semantically meaningful regions such as rooms or hallways could be particularly effective for CPP algorithms where some higher-level optimizations of the coverage plan are desired. Using these semantic cell decompositions, coverage plans could more easily be designed to prioritize coverage of room cells or use sparser coverage patterns in hallways cells. A detailed discussion of semantic map segmentation is outside the scope of this work but the survey of Bormann et al. provides a comprehensive discussion of the most popular methods [10].

As we transition our discussion to the next class of CPP methods, it is worth noting a tendency of the research we have discussed to focus on the decomposition method rather than generating the coverage path within each cell. For a large number of these algorithms, the cell area is often described as being ”easy” to cover with simple back-and-forth or spiraling motions without providing specific implementation details, although there are several notable exceptions. In their work on CPP for agricultural applications, Oksanen and Visala provide coverage path examples that take into consideration the vehicle turning dynamics in between concurrent back-and-forth passes and describe a procedure for generating spiral-like coverage paths using techniques from computational geometry such as polygon offsetting [72]. As part of their research into optimal coverage path planning using genetic algorithms, Jimenez et al. use variations of two different ”motion templates” to generate feasible coverage patterns for each cell following decomposition [40]. In the next subsection we further shift our focus towards methods for generating actual coverage paths as we examine global approaches to CPP where the target area is considered as a whole.

2.1.3 Global Methods

Continuing along the decomposition spectrum we finally arrive at the global CPP methods, which operate on the entire target region without the need to decompose it into any sort of grid or cells. Although they don't specifically require decomposition, these methods can generally also be applied to each cell within a cellular decomposition to determine a coverage path for that cell. These global methods include both heuristic and complete approaches, the former of which do not guarantee that the entire target environment will be covered.

Perhaps the most popular of the heuristic methods is random coverage. This approach is particularly common among many simpler floor cleaning robots due to its ease of implementation and basic sensor requirements. Using this technique, the robot moves around the target area randomly, turning whenever an obstacle is encountered or at some random interval. A disadvantage of this method is that as the domain increases in size or complexity, it can become less effective; environmental features and geometry may prevent the robot from entering specific areas unless entering from exactly the right angle or a certain area may lend itself to being consistently missed by the robot.

In their work, Palacin et al. developed a computer vision system for measuring the performance of cleaning robots executing random coverage [73]. The system makes use of a camera mounted over the target scenario to acquire imagery of the scene and then processes the images with a background subtraction technique to track the position of the robot over time. By analyzing the results of several experiments, the authors are able to show an exponential relationship between the coverage percentage and the operating time for a circular cleaning robot moving in random directions after reaching a wall in a rectangular, obstacle-free target area. This result perfectly illustrates that a simple random coverage approach can quickly cover most of an open area but achieving complete coverage is unlikely no matter how long the robot runs.

Clearly, a random method is not the most suitable technique for high-risk applications where missing a certain section of the target area could have drastic consequences, such as structural inspection or demining. If the sensor being used to cover the target area is imperfect, however, this can change the calculus of whether a random approach is appropriate. In his work Gage evaluates the performance of random coverage in the context of demining with imperfect sensors that have a detection probability of less than 1.0 [29]. In this case, even provably complete coverage cannot guarantee certain success and in fact, as this detection probability decreases, the advantages of complete systematic coverage over random coverage diminish.

In order to improve some aspects of random coverage methods, there has been some research into combining these techniques with complete coverage path planning. In their work, Liu et al. propose a hybrid approach that alternates between random movement and a boustrophedonic complete coverage path [58]. This strategy seeks to benefit from some of the advantages of each method, using random paths to more rapidly explore the target region and mitigate potential localization issues that may impair complete coverage methods, while relying on the latter to improve the rate of coverage over time.

As far as complete global CPP methods go, by far the most common approaches are to implement a back-and-forth boustrophedonic or contour-following spiral pattern. These techniques have their roots in early work on tool path generation for pocket machining, where they are referred to as direction-parallel and contour-parallel milling respectively [35]. While the direction-parallel approach has been the most widely adopted fill pattern by today's 3D printers due to its simplicity [32], the boustrophedonic pattern results in many sharp turns, which can be problematic for additive manufacturing and robotic coverage applications alike; as a tool or robot reaches one of these sharp corners, it must decelerate in one direction and then accelerate in another direction. Contour-parallel paths are comprised of a set of closed curves parallel to the perimeter of the target region and generally result in smoother turns and object boundaries than direction-parallel methods [25,116]. However, if the target area has a more complex shape, these methods can lead to isolated "pockets," which are disconnected and result in path plurality. In their work, Jin et al. propose a hybrid fill pattern that generates a few contours before filling the remaining area with a direction-parallel pattern [41].

Interestingly, while research into CPP drew extensively from work on tool path generation in its early days, this crossover faded somewhat over time, perhaps due to the different optimization criteria for manufacturing and robotic coverage applications. The Hilbert space-filling curve gained some popularity as a means of generating tool paths for additive manufacturing due to its good preservation of locality, meaning two points that are close within the 2D space being filled are also close along the Hilbert curve. This property is advantageous for additive manufacturing because the bonding quality between two adjacent lines of material can be adversely affected if one is allowed to cool and shrink before the other [8]. On the other hand, this locality property is less important for most applications of robotic coverage and the high number of corners on a Hilbert curve can be difficult to follow effectively for a mobile robot. Additional fractal-like space filling curves have been suggested by Wasser et al. in their work on tool path generation for Fused Deposition Modeling (FDM) [107]; however, these methods likewise feature numerous sharp corners, which limit their use for CPP.

Recently a new kind of space-filling curve has been developed that shows promise as a potential CPP technique as well. In their research, Zhao et al. introduce the connected Fermat spiral, which in contrast to classical patterns like the Hilbert curve, are made up of mostly long and low-curvature paths [119]. In addition, it is generally possible to fill a region with a connected Fermat spiral such that the start and end points of the curve lie next to each other on the outside boundary of the target region. This property allows for convenient joining of multiple spirals to fill groups of connected regions with ease. Both this property and the low-curvature nature of the curves, make this approach more suitable for robotic coverage applications.

2.2 Energy-Efficient Path Planning

Due to the inherent power constraints of mobile robots, energy-efficiency has been a continued area of interest in robotics, motivating extensive study and engineering efforts. As aerial drones with strict payload limitations rapidly gain popularity and robots are increasingly deployed in remote scenarios from the ocean depths to other planets, the need to understand robotic power consumption and develop energy-saving algorithms and technology that increase the lifespan and utility of these vehicles has intensified and garnered further attention. Mei performed a comprehensive study of robot power usage in the context of common robotic subsystems such as locomotion, sensing, communication, and computation, and developed several strategies for energy-efficient motion planning, exploration, and deployment [64,66]. While continued progress in power-efficient electronics has helped reduce the energy footprint of sensing, communication, and computation, motion remains a major power sink for robots [61]. Although energy-efficiency has historically taken a back seat in path planning research, which focused on providing safe, goal-oriented, and/or time-optimal paths [50,52,69,86,90,102], energy-efficient planning techniques have emerged as a means of mitigating the energy costs of robotic locomotion and have recently become the subject of renewed interest.

Energy-efficient planning methods have been suggested at all levels of the autonomy hierarchy, from high-level task schedulers, to the low-level controllers guiding basic motion. At the topmost abstraction level of autonomy, energy-efficiency is achieved through dynamic power management and intelligent scheduling of tasks according to their power requirements, available energy, and mission objectives. In practice this approach involves selectively modulating or disabling idle robot subsystems and timing execution of power-hungry operations to occur when their cost is least or energy is abundant; for example, a ranging sensor's sampling rate can be reduced when the robot is moving slowly, communications can be scheduled for when transmission distance is shortest, and uphill locomotion can be limited to when solar power is available [65,67,81]. In order to apply these techniques successfully, the available knowledge of vehicle power consumption and operation, domain-specific sources of energy, and overall mission objectives is typically encoded in some representation that can then be used to produce valid schedules for vehicle actions in the given scenario. Since task scheduling is a well-studied problem in computing, common representations such as the constraint graph are often used to allow for convenient adaptation of existing scheduling algorithms to robotics applications [17,49].

One example that illustrates the potential of high-level task scheduling to improve energy-efficiency is the work of Liu et al., which describes a power-aware scheduling technique that can satisfy typical time and power constraints, while additionally maximizing the use of available free power [57]. The researchers present their methodology in the context of the NASA/JPL Mars Pathfinder rover, which used a solar panel and non-rechargeable battery as its power sources and was designed with a low-power task scheduler that enabled the rover to operate for hundreds of days during daylight. By extending the constraint graph problem formulation and discerning the free power provided by the solar panel from the costly power sourced from the battery, their scheduling technique is able to achieve significant improvements in execution time and energy usage over the system implemented on the Mars rover. Although such high-level approaches towards energy-efficiency can be very effective, detailed constraint graph representations become increasingly difficult to define as the complexity of the robot or mission increases, limiting the scope of their use.

To bridge the gap between task scheduling and path planning, Tompkins, Stentz, and Whittaker pioneered the mission-level planning approach in their work creating TEMPEST, a planner that combines knowledge of mission objectives, operational constraints, rover performance, and the environment to solve the coupled path planning and resource management problem [100]. Initially developed in the context of long-duration, solar-powered, planetary exploration in polar regions, TEMPEST was first used to guide a solar-powered rover using a sun-synchronous navigation strategy [109,110] and knowledge of available solar energy and anticipated power draw. In this implementation, TEMPEST runs offline prior to rover operations and computes mission plans that combine time-sequenced path plans, battery energy guidelines, and a schedule for battery recharge intervals in order to achieve energy-efficient rover operation. Deployed on the solar-powered Hyperion rover in the field, the plans supplied by TEMPEST enabled continuous operation for 24-hour time periods, during which the rover executed multi-kilometer traverses while maintaining substantial battery energy reserves for the duration. In their follow-up work, Tompkins, Stentz, and Wettergreen extended TEMPEST to run online, support additional rover actions, and use re-planning to repair plans during execution, and present results from the field [99,101]. Wettergreen at al. further deployed TEMPEST on the Zoë rover during multiple field experiments in the Atacama Desert involving a total of 55km of autonomous navigation [108].

Since the energy consumption of an electric motor used to propel a typical mobile robot fundamentally relies on two parameters, speed and load, approaches to improve energy-efficiency at the path planning abstraction level generally seek to optimize the velocity of the robot along its trajectory or search for advantageous routes through the domain that balance travel time and motor load. Given an arbitrary input trajectory for a robot to follow, Tokekar et al. use closed-form solutions to compute optimal velocity profiles with an optional constraint on maximum velocity and develop a dynamic programming scheme to extend their method to chains of input trajectory segments [96, 97]. Wang et al. take a similar dynamic programming approach in their work to optimize mobile sensor placement, but additionally consider scenarios when road conditions, and thereby load, are not uniform along the path [106].

Instead of trying to optimize velocity along a given path, Sun and Reif study the problem of finding paths over terrain that minimize energy expenditure by a robot due to gravity and friction [93]. Adapting their previously developed BUSHWHACK algorithm, they are able to approximate energy-efficient paths across the terrain that inherently take advantage of friction and gravity (e.g., zig-zagging up steep slopes) [82,92]. In our previous work on energy-efficient planning through flow fields, we build upon a planner developed by Lolla et al., which uses level set evolution to compute time-optimal paths for a vehicle moving at a constant nominal velocity through a completely known flow field [59]. By considering the dynamics of the ASV we developed for our research [85,103], we predict the velocity that will be attained by the robot for initial path planning and then periodically trigger re-planning using the true velocity that was achieved [98]. Using this approach, we were able to achieve some improvement in simulation, but have not been able to test in the field due to a lack of complete surface current information, on which this method is strictly dependent.

At the lowest level of the autonomy hierarchy, some groups have approached the task of achieving energy-efficiency for their robots by encoding some sort of behavior in the control laws governing their vehicles' operation. In their work, Kim and Kim use optimal control theory to compute minimum-energy translational velocities along a straight line for differential drive robots and achieve energy savings of up to 10% over other energy-optimizing methods [48]. In their work applied to autonomous sail boats, Sauze and Neal use a biologically inspired control technique to achieve significant energy savings during operation [83, 84]. Other work done by Derenick, Michael, and Kumar similarly takes a control-centric approach to achieve energy-aware behavior with a team of quadrotors, but focused on maintaining continuous coverage of the region as agents leave the area to dock and recharge [18].

Although energy-efficient operation can successfully be achieved with techniques across the autonomy hierarchy, most approaches rely on exploiting domain knowledge and accurate system models, which are often incomplete, unavailable, or difficult to obtain outside of simulation. Further difficulty arises from the fact that environmental conditions and vehicle dynamics can change significantly between deployments or during the course of operation, which can reduce the effectiveness of energy-efficient methods even to the point that energy consumption is increased. These challenges have conspired to limit the widespread adoption of such techniques in the field and motivated efforts to develop approaches that can adapt to changing conditions and balance domain exploration to refine environmental knowledge with the exploitation of this information to improve energy-efficiency.

In their work with UAVs, Lawrance and Sukkarieh develop a method to simultaneously map and utilize a dynamic wind field using soaring flight in order to significantly extend UAV flight duration [53]. Wind speed and direction at the vehicle location are measured using a combination of air data and inertial measurement sensors and then used as input to a Gaussian process regression to estimate a wind map. Their planner then makes use of this wind map and aerodynamic models of the UAV to determine new target locations that will allow for data collection resulting in wind map improvements and paths to these targets that best capture the available ambient wind energy. By balancing exploration and exploitation of the wind environment, their planner is able to achieve consistent energy gain in simulations with dynamic wind fields.

Since environmental conditions such as ambient winds and currents are often mercurial, even domain knowledge gathered during exploration can be highly uncertain. A common approach to overcome this uncertainty is to dynamically replan and adjust models during operation as new information is collected and prior knowledge is invalidated. An example of this technique is demonstrated by Jones and Hollinger in their work to develop a framework for planning energy-efficient trajectories for a vehicle moving through uncertain environmental disturbances [45]. Their approach generalizes previous trajectory optimization techniques, such as STOMP [46], by removing the assumption that trajectory waypoints are equally spaced in time, thereby allowing the optimization to vary the temporal length of the trajectory. This strategy enables their algorithm to optimize trajectories over a wider range of cost functions, which can be specifically defined to plan energy-efficient paths.

In order to address environmental uncertainty, Jones and Hollinger additionally build a model describing deviations from the forecast disturbances using a GP regression and observations collected by the vehicle, which is used to regularly replan new potential paths.

The energy costs of newly computed trajectories are then compared to that of the existing path being executed and can be selected to replace the current path depending on user-selected parameters. Experiments in simulation reveal that the choice to accept a replacement path critically affects the performance of this framework; metrics which estimate and weigh the probability that the replacement path consumes less energy are shown to improve energy-efficiency on average while greedy switching actually leads to worse performance. Jones and Hollinger further validate their framework with a series of field tests using the ASV platform we developed in our previous work [85,103] deployed on a lake in windy conditions. By selecting an appropriate replanning strategy, the system is able to reduce the energy consumption during a traverse of the lake without any a priori knowledge of the wind patterns across the domain.

2.3 Energy-Efficient Coverage Planning

While energy-efficient planning has shown continued promise when applied to planning problems where the robot is guided between a subset of points within the domain, problems such as complete coverage planning provide additional structure that presents an opportunity for further exploitation to achieve even greater performance. In this section we examine the intersection of the coverage planning and energy-efficient planning methods from our previous discussion.

One approach to energy-efficient coverage planning involves splitting up a coverage path to introduce stops where a robot can refuel or recharge prior to continuing its work. This kind of servicing behavior was implemented by Oksanen and Visala in their work on coverage for agricultural machines, but was considered in the context of spreading and harvesting operations where the robot must make stops to fill or empty its tanks respectively, rather than vehicular energy constraints [72]. In order to achieve this, their algorithm simulates the tank level along a coverage path to determine whether the tank ever reaches a critical level requiring servicing. If such a point is identified, the coverage path is modified to visit a servicing location before the critical level is reached, after which the simulation is repeated to check whether additional service stops are required.

Strimel and Veloso specifically tackled the version of this problem where the robot covering the target area has finite energy resources [91]. Building on the boustrophedon cellular decomposition coverage algorithm, a new sweeping planning algorithm is introduced that augments the cellular decomposition with an additional cell to represent the service or charging station. After a tour of cells is determined, it is then optimally partitioned into subroutes which begin and end at the charging station and meet the energy constraint of the robot. These subroutes are then appended together to determine the shortest coverage plan throughout the region while obeying energy constraints.

Although this type of servicing behavior is a practical solution to achieving complete coverage when energy is limited, the coverage paths generated are not necessarily optimal and there are aspects of these paths that can be modified to have a significant effect on execution costs. In their work on optimal coverage planning for arable farming, Jin and Tang consider several of these aspects, most notably the cost of various turning techniques between the straight transects that make up typical back-and-forth boustrophedonic coverage patterns [42]. While it is a common approach to choose the direction of boustrophedon coverage corresponding to the longest dimension of the target area since this can minimize the number of required turns, the authors demonstrate that the incident angle between the transects and the edges of the target region can substantially affect the total length of the coverage path as well. Since the energy required to drive a path at a given speed generally increases with the length of the path and sharp turns are expensive due to the need to decelerate then accelerate the vehicle, optimizing the turns along the coverage path can have a profound impact on its energy cost.

As additional domain knowledge becomes available to the planner, this opens up various avenues for further optimization. 3D terrain information has become a popular choice to leverage in order to improve coverage path planning performance due to its availability and broad utility across a variety of applications. Jin and Tang utilize terrain knowledge in order to extend their prior work optimizing turns along coverage paths to three dimensions, while introducing additional path optimizations that improve coverage rate and decrease soil erosion [43].

In their work, Dogru and Marques specifically consider the task of energy-efficient coverage planning for ground robots when terrain of the survey environment is known [23, 24]. They propose an alternative decomposition of the environment into regions with similar terrain slope and use a genetic algorithm to optimize the orientation of the boustrephedon path computed in each region and the order in which they should be visited by the robot in order to minimize energy usage. The proposed technique is shown to successfully reduce the energy requirement to cover several scenarios in simulation; however, it assumes a constant vehicle velocity, which prevents exploitation of the potential energy of the vehicle when descending from a higher region.

More recently, Wu et al. introduced an energy-efficient coverage path planning algorithm for 3D surfaces [114] that builds on the Fermat spiral tool path generation technique originally proposed by Zhao et al. [119]. By modifying the geodesic field used when the spirals are computed and applying a heuristic to minimize height variation along the coverage path, their algorithm is able to plan paths with significantly reduced energy costs when compared to other coverage algorithms for 3D terrain. Similarly to Dogru and Marques, the authors assume braking when descending from higher terrain, which again leaves some energy optimization on the table.

Beyond knowledge of the terrain throughout the domain, coverage planners can take advantage of other sources of ambient energy within the environment, such as solar radiation. In her work on coverage planning for lunar rovers, Shillcutt investigates the effect of the coverage pattern used on the energy cost to completely survey an area in the context of solar-powered lunar rover exploration [87,88]. By taking into account orbital ephemeris and terrain data as well as models for solar power generation by the robot, Shillcutt was able to evaluate the energy cost of specific coverage patterns and develop sun-synchronous and sun-seeking algorithms to extend rover lifetime and increase mission productivity. The coverage pattern energy cost evaluation was validated with data collected during tests of autonomous searches for meteorites in the Antarctic [4,5] while performance of the sun-synchronous and sun-seeking behaviors was evaluated in simulation.

In their work, Vasisht and Mesbahi address a similar problem of energy-aware coverage planning in the context of persistent monitoring of a geographical area for events of varying priorities by a UAV equipped with solar cells [105]. By building a model of the energy collected by the solar cells on the UAV which depends on the ambient conditions and vehicle orientation, the authors can estimate the net energy cost of moving between points within the survey region and compute energy-optimal trajectories. The coverage planning algorithm developed then makes use of this model to maximize coverage time of the area while prioritizing visits to regions that not been observed for the longest time. Using this approach, the authors are able to extend the total persistent coverage time by the UAV in simulation.

2.4 Summary

In this chapter, we reviewed a subset of the available literature and research that addresses coverage path planning and energy-efficient path planning for autonomous vehicles. While there are a multitude of methods that tackle these problems individually, the existing work targeting the combination of these topics is strikingly limited by comparison. As outlined in the previous section, the primary approaches to energy-efficient coverage planning involve breaking up coverage paths with stops to service (e.g., refuel or recharge) the vehicle, optimizing the number and types of turns along the path, considering terrain information to design more efficient paths, and incorporating knowledge of ambient solar radiation to improve collection of energy during execution. The absence of planning methods that apply other established techniques of conserving energy to the coverage planning problem across a broader set of domains motivates the work in this thesis.

In our discussion of energy-efficient path planning, we examine a variety of methods ranging from strategic task scheduling to velocity optimization along a trajectory, which would seem like perfect candidates for adaptation to coverage planning. For example, intelligently scheduling the power-hungry suction motor on a robotic vacuum to turn off when traversing regions that have already been cleaned could have a significant impact on the coverage range of the unit. Similarly, for automated surveys of large outdoor areas, sensors and radios could be powered down when passing through a visited region or out of communications range respectively. Adjusting drive speed to let a vehicle coast down a decline also seems like a natural approach to save energy and yet existing energy-efficient algorithms for coverage of 3D terrain often assume a fixed velocity along the coverage path. Although they may be simply stated, actually implementing these strategies in a coverage path planner can be complicated and typically involves developing a custom, application specific solution that might build on a general CPP approach - an effective deterrent for further research into this important problem.

The goal of the novel coverage path planning framework introduced in this thesis is to remove this barrier and offer a clear process for quickly building coverage planners and experimenting with various optimizations without the need to rewrite significant parts of an algorithm. Using this framework, optimizations such as intelligent throttling of high power devices and decreasing thrust depending on the local environment can be implemented as standalone components that can be applied to existing coverage planners independently or in a chain, which can result in synergistic benefits that are unexpected or would have been difficult to design. Moreover, this framework allows for further reasoning later in the planning process based on any applied optimizations. This enables the design of complex behavior such as coverage where communications radios are turned off when out of range, but the longest stretch of time the vehicle spends out of range is minimized. In the next chapter, we present our framework and demonstrate how it can be used to implement several common coverage planners.

Chapter 3

Constraint-Based Coverage Path Planning

In this chapter, we present the Constraint-Based Coverage Path Planning (CB-CPP) architecture and describe the key stages within this planning pipeline. We begin by introducing the concept of a Path Constraint and offer some simple examples of how these might be implemented. Next we describe the process by which path constraints can be manipulated and assembled into a coverage path. Finally, we present several case studies in which we walk through the implementation of common CPP algorithms within our CB-CPP framework and conclude with a discussion of the advantages of planning within the framework.

3.1 Path Constraints

A path constraint is a set of details that limits different aspects of the complete path being constructed. In its most basic incarnation, this boils down to a pure spatial constraint, which simply represents a set of locations that must lie on the final path determined by the planner. By designedly laying out a set of such constraints throughout the target region and then chaining them together appropriately, it is possible to build path plans with a variety of characteristics for different applications, including notably complete coverage paths.

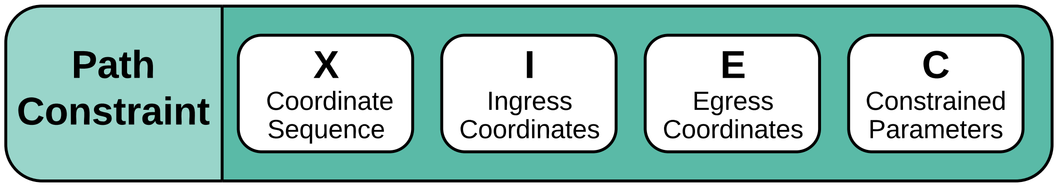


Figure 3.1: Formal path constraint definition

To this end, we formally define a path constraint as an ordered set $P = (X, I, E, C)$ where X is a sequence of coordinates, I and E are sets of possible ingress and egress coordinates respectively, and C is a set of additional constrained parameters. When referring to a specific path constraint P_i, we adopt the graph style notation $X(P_i)$, $I(P_i)$, $E(P_i)$, and $C(P_i)$ to refer to these components, while the set of all path constraints is referred to by $\mathbb{P}$.

Each component of this definition serves a specific purpose in the context of assembling constraints into a complete path to achieve some goal. The coordinate sequence X localizes the constraint, both providing the actual coordinates that will ultimately make up part of the complete path and coupling the parameter constraints defined in C to particular segments along that path. The ingress points I and egress points E facilitate the linking of multiple path constraints into a single path by informing planners of valid points at which to begin and end following a constraint. Finally, the constrained parameters defined in C describe further restrictions to a vehicle's operation as it moves along the the constraint.

Before delving into the roles of each path constraint component in detail with some examples, it is helpful to first partition the set of all possible path constraints by whether their defining geometry is open or closed. We now discuss each of these partitions individually, providing basic examples and illustrating important differences that should be taken into account during implementation.

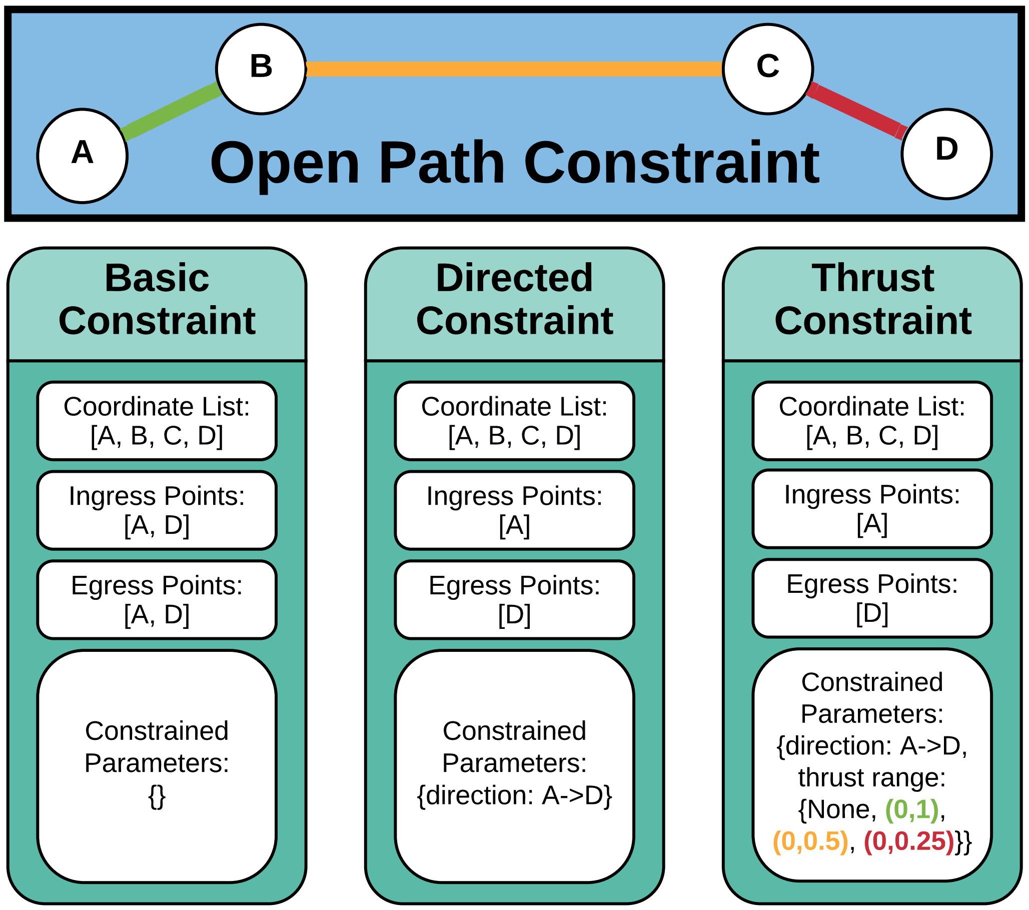
3.1.1 Open Path Constraints

Conceptually, open path constraints are just path constraints defined by open geometry such as sequences of line segments or curves. Since these geometries can intersect themselves and even begin and end at the same point, however, the primary defining characteristic of constraints in this subset is that they can have a maximum of two possible ingress and egress points because their defining coordinate sequence must be executed in its entirety, either forwards or in reverse. Using this fact we can formally define the open path constraint subset as follows: $\mathbb{P}_{\mathbb{O}} = \{P_i : P_i \in \mathbb{P} \text{ and } |I(P_i)| \leq 2 \text{ and } |E(P_i)| \leq 2\}$.

Like all path constraints, each open constraint definition includes a coordinate sequence X that contains the coordinates of all points along the defining geometry. The sets of valid ingress and egress points, I and E respectively, may both include the first and last elements of a constraint's coordinate sequence X if the constraint is undirected and can be executed in either direction, or may each contain a single point if the constraint is directed and must be followed in a specific direction. Examples of both undirected and directed open path constraints are shown in figure 3.2.

In addition to its coordinate sequence and ingress/egress points, each path constraint should define a set of parameter constraints C, which can restrict additional aspects of movement along the constraint to achieve specific goals. In the most basic example of an undirected open constraint, this set is empty, but in the directed example, a direction is imposed on the constraint using an entry in C, thereby specifying single ingress and egress points that define the final execution order of the coordinate sequence X. This type of parameter constraint is universal, meaning it applies to the entire constraint, but more advanced parameter constraints that change along different segments can also be specified by defining a sequence of values that matches the length of the coordinate sequence X.

An example of this type of varying parameter constraint is given by the Thrust Constraint in figure 3.2. This particular instance limits the thrust a vehicle can apply as it moves along the constraint by specifying an allowable thrust range for each point in the constraint. Using this representation, it is possible to remove the parameter constraint from sections of the constraint altogether by specifying a null/none value for those points. In the example shown



A
B
C
D
Open Path Constraint
Basic Constraint
Coordinate List: [A, B, C, D]
Ingress Points: [A, D]
Egress Points: [A, D]
Constrained Parameters: {}
Directed Constraint
Coordinate List: [A, B, C, D]
Ingress Points: [A]
Egress Points: [D]
Constrained Parameters: {direction: A->D}
Thrust Constraint
Coordinate List: [A, B, C, D]
Ingress Points: [A]
Egress Points: [D]
Constrained Parameters: {direction: A->D, thrust range: {None, (0,1), (0,0.5), (0,0.25)}}

Figure 3.2: Examples of open path constraints with varying levels of specificity

in figure 3.2, there is no constraint on thrust the vehicle can use to reach waypoint A. For the subsequent waypoints, the allowable thrust is increasingly restricted from a range of 0 to 1.0 for waypoint B, to 0 to 0.25 for waypoint D. Other possible parameters to constrain include velocity ranges, vehicle orientations, and sensor/actuator state, which can be used to enforce safe velocities in critical areas, optimize environmental effects (e.g., orient vehicle to decrease drag), or reduce energy usage due to extraneous sensing/actuation (e.g., turn off vacuum when travelling over already visited regions).

3.1.2 Closed Path Constraints

In order to facilitate the implementation of planning algorithms that make use of contours or offset polygons, it is convenient to be able to construct path constraints defined by closed geometry. Closed path constraints provide this capability through a slight modification to their implementation that uses their coordinate sequence X to specify the vertices of a closed polygon, the boundary of which then becomes the defining geometry for the constraint. In contrast to open constraints, closed constraints will always share the same ingress and egress points, or more formally $I(P_C) = E(P_C)$ for all closed path constraints P_C. Although it is feasible to implement closed path constraints in such a fashion as to allow ingress and egress at any point along the closed boundary of the polygon defined by X, we limit our implementation to allow ingress/egress solely at the coordinates defined in X, that is $\forall i \in I(P_C), i \in X(P_C)$ and $\forall e \in E(P_C), e \in X(P_C)$.

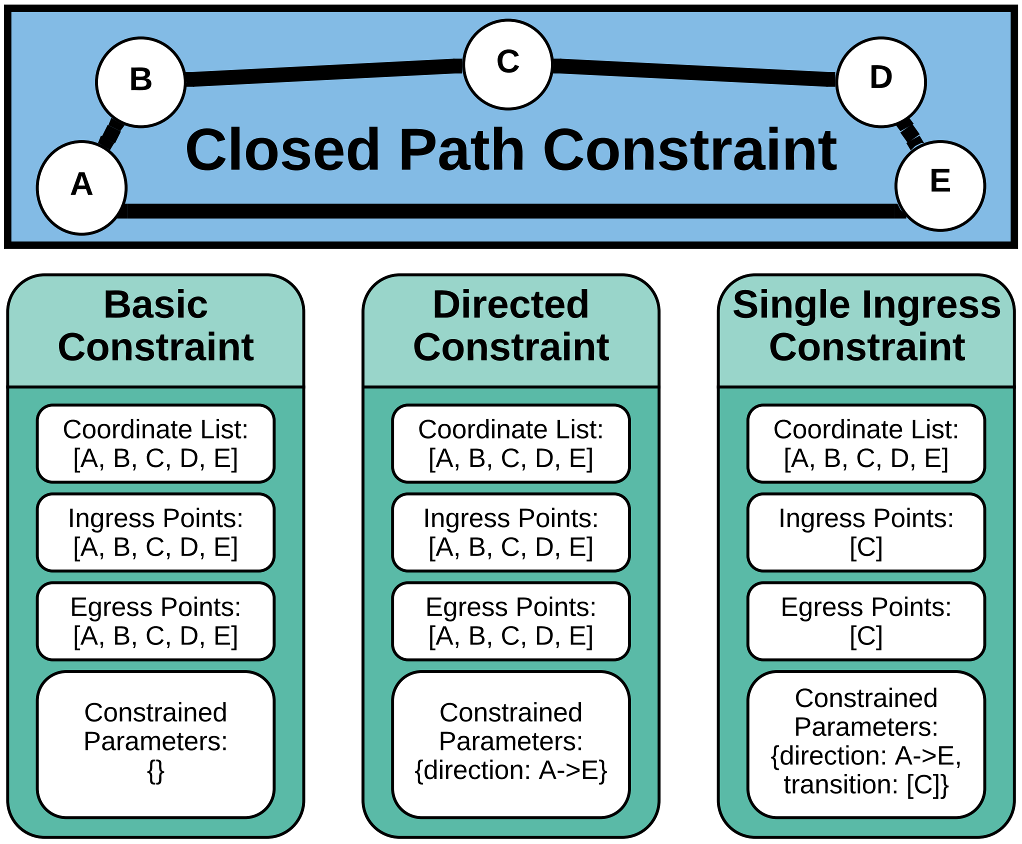

Closed Path Constraint
A
B
C
D
E
Basic Constraint
Coordinate List: [A, B, C, D, E]
Ingress Points: [A, B, C, D, E]
Egress Points: [A, B, C, D, E]
Constrained Parameters: {}
Directed Constraint
Coordinate List: [A, B, C, D, E]
Ingress Points: [A, B, C, D, E]
Egress Points: [A, B, C, D, E]
Constrained Parameters: {direction: A->E}
Single Ingress Constraint
Coordinate List: [A, B, C, D, E]
Ingress Points: [C]
Egress Points: [C]
Constrained Parameters: {direction: A->E, transition: [C]}

Figure 3.3: Examples of closed path constraints with varying levels of specificity

Several examples of closed path constraints are shown in figure 3.3. In the simple case when a closed path constraint P_C is virtually unconstrained beyond its defining geometry, the sets of ingress and egress points $I(P_C)$ and $E(P_C)$ are congruent to the set of defining polygon vertices $X(P_C)$. Like open path constraints, we can enforce a direction of travel along the constraint by adding an entry in the set of constrained parameters $C(P_C)$, as shown in the directed constraint in figure 3.3. Unlike open path constraints, however, defining a direction for the closed constraint does not affect its sets of possible ingress and egress points.

In order to fully constrain a closed path constraint such that it can be represented by a sequence of coordinates, we must define a transition point, which represents the point at which an agent can begin executing the constraint. Defining such a point will collapse the sets of valid ingress and egress points to a singleton set containing the transition point, indicating that the constraint must begin and end execution at this point. An example of such a fully defined closed path constraint is given in figure 3.3.

Once a closed path constraint has a defined transition point, it can be easily be converted into an open path constraint with the same ingress and egress point. This feature can enhance code reuse and simplify adapting algorithms built for open path constraints to work with closed path constraints.

3.2 Constraint-Based Coverage Planning

By introducing the concept of a path constraint as a building block of coverage paths, we laid the foundation for designing coverage planners that can reason about the order or direction in which constraints are executed, and take into consideration available domain knowledge encoded in additional constraint parameters. In this section we present a general framework for such a constraint-based coverage planner.

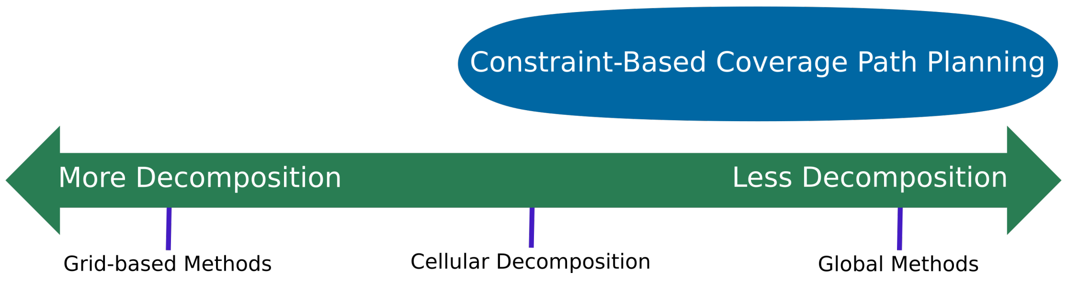
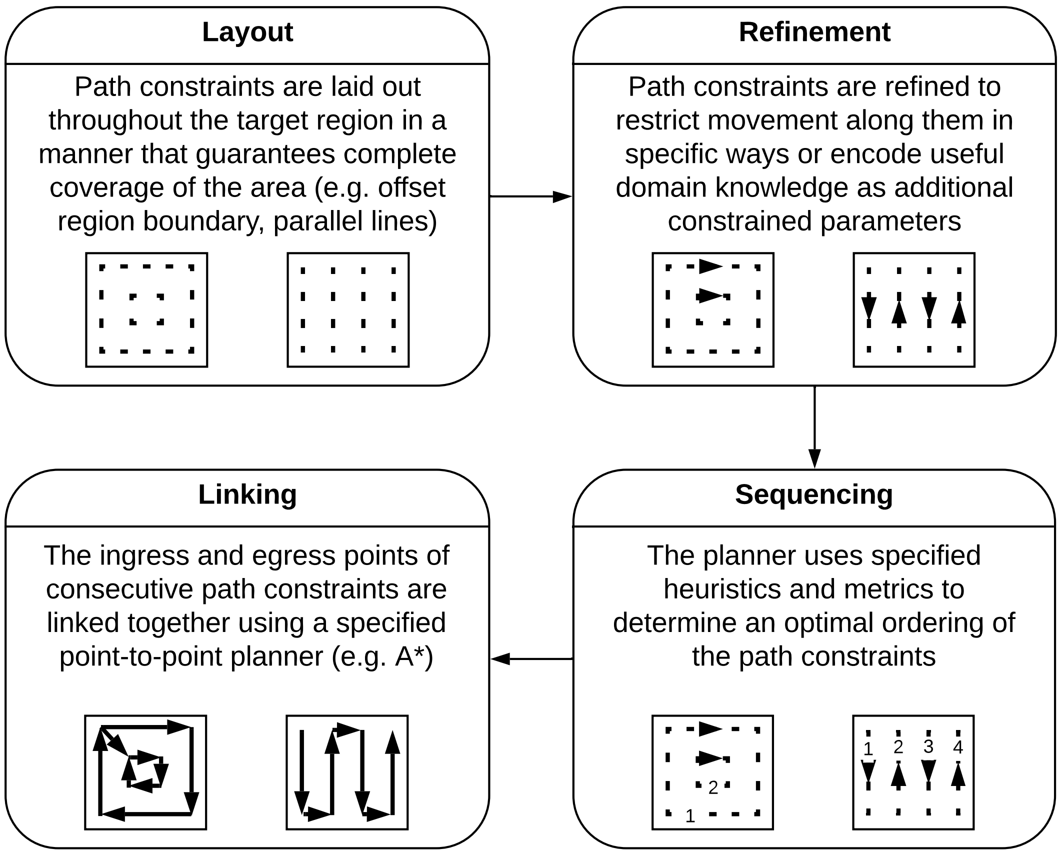


Figure 3.4: Where CB-CPP methods fall on the Decomposition Spectrum

When considered in the context of our discussion of existing coverage path planners in the previous chapter, CB-CPP implementations naturally fall into the global methods section of the decomposition spectrum because they do not themselves contain a method of decomposing the target coverage area. They can, however, be readily combined with any number of existing decomposition schemes to build coverage paths for each cell, which can then be assembled into a complete coverage path. In fact, knowledge of the ingress and egress points into and out of each cell can easily be fed into a CB-CPP algorithm from the decomposition in order to further optimize the coverage path constructed. Moreover, additional knowledge from the decomposition may be useful during the different phases of CB-CPP, so a more deliberate coupling of decomposition and CB-CPP algorithms can yield further benefits.

Our framework for constraint-based coverage planning consists of four distinct stages: Layout, Refinement, Sequencing, and Linking. When designing a coverage planner using this framework, the developer can choose an existing implementation for each component or implement a custom component provided that the interfaces defined for each stage are respected. Typical constraint-based planner implementations will have one layout component, any number of refinement components, one sequencing component, and one linking component.



Layout
Path constraints are laid out throughout the target region in a manner that guarantees complete coverage of the area (e.g. offset region boundary, parallel lines)
Refinement
Path constraints are refined to restrict movement along them in specific ways or encode useful domain knowledge as additional constrained parameters
Sequencing
The planner uses specified heuristics and metrics to determine an optimal ordering of the path constraints
1
2
1
2
3
4
Linking
The ingress and egress points of consecutive path constraints are linked together using a specified point-to-point planner (e.g. A*)

Figure 3.5: Stages of the constraint-based coverage planning framework

3.2.1 Layout

The layout component is responsible for distributing the path constraints throughout the target region in a manner that ensures complete coverage if a robot executes all the constraints.

An important consideration when designing a layout component of a coverage planner is the footprint of the target vehicle as well as its sensor or actuator payload. The vehicle footprint defines a safe envelope surrounding the robot, which specifies a boundary that may result in a collision if breached. The sensor or actuator footprint, hereby referred to as the payload footprint, defines the reliable range of the sensor or actuator mounted on the vehicle (e.g., field of view and range of a sonar sensor or inlet shape and position on a vacuum). As shown in figure 3.6, the geometry and configuration of these footprints can vary in size and shape: the vehicle footprint can be smaller than, larger than, or congruent to the payload footprint, and the footprints can be of different shapes and asymmetric.

The choice of vehicle and payload footprints can have profound implications on the coverage quality of the final path and complexity of the planning algorithm, particularly relating to the layout of path constraints. In fact, if the payload footprint is completely contained within the vehicle footprint and their boundaries do not touch, complete coverage of any specified target area is not possible without breaching either its boundary or the

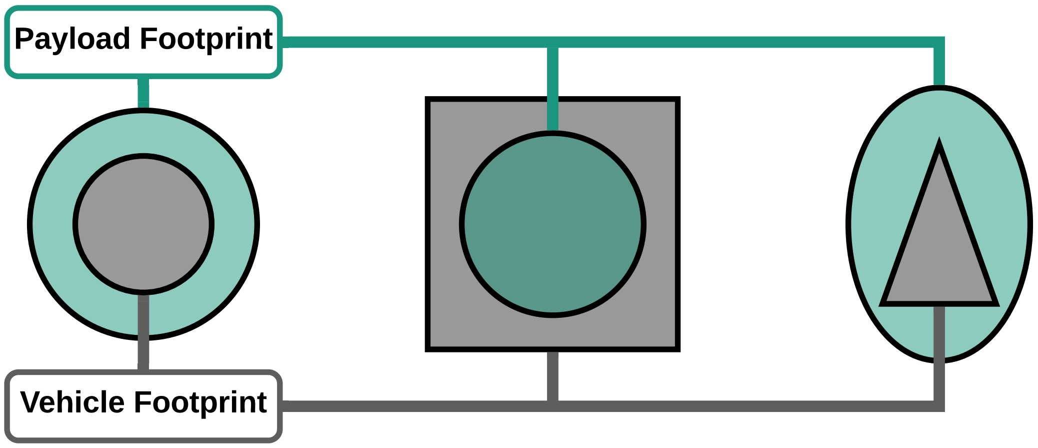


Figure 3.6: Vehicle and Payload Footprints can be of Different Shapes and Sizes

vehicle footprint. Asymmetric footprints pose an additional challenge as vehicle orientation must be accounted for during constraint layout. If the target vehicle is holonomic, the desired orientation can be encoded on each path constraint as an additional constrained parameter, however, for non-holonomic vehicles this may result in path constraints that are impossible to execute for the target vehicle. Perhaps the simplest approach is to define both vehicle and payload footprints with congruent symmetric polygons if appropriate, although this can often reduce the coverage quality if the payload footprint is overestimated and leave potential efficiency on the table if the vehicle footprint is excessive. Since the specifications for layout components within the CB-CPP framework do not restrict the footprint configuration that can be used, this choice is left up to the developer to balance the the tradeoff between improved coverage and implementation complexity according to the needs of his application.

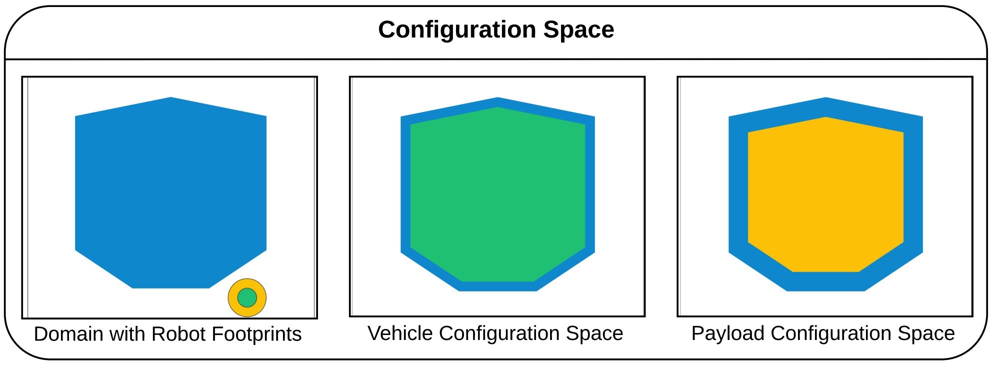

Configuration Space
Domain with Robot Footprints
Vehicle Configuration Space
Payload Configuration Space

Figure 3.7: Choice of Vehicle and Payload Footprints Affects Configuration Space

In addition to the vehicle and payload footprints, the geometry of the target region plays a significant role in the constraint layout process. In order to layout constraints, the planner must determine how close to the boundaries of the target area and any obstacles within the vehicle should drive to avoid collisions while still achieving full coverage of the region. A common approach is to first determine a configuration space for the robot and the given survey area according to the vehicle and payload footprints. This space defines a safe region within which constraints can be laid out in a systematic fashion without concern of collision or departure from the target area. Two different configuration spaces determined using a sample domain and robot footprint specification are show in figure 3.7.

Once a suitable configuration space has been determined, there are a number of strategies for laying out constraints to cover this space. One approach that will produce a set of parallel line constraints throughout the region is the line sweep approach illustrated in figure 3.8. This method relies on moving a line that extends beyond the boundaries of the configuration space along a specified sweep direction in fixed increments. After each movement, the intersection points of the sweep line with the configuration space polygon are used as the coordinates of a new path constraint. This process is analogous to the line sweep methods used for cellular decomposition by several common coverage path planners [14,51,113] and the resulting straight line path constraints are well-suited for implementing the classic boustrophedon coverage pattern. The distance the sweep line is advanced at each step can be determined by calculating the number of transects required to fully cover the region depending on the robot's payload footprint.

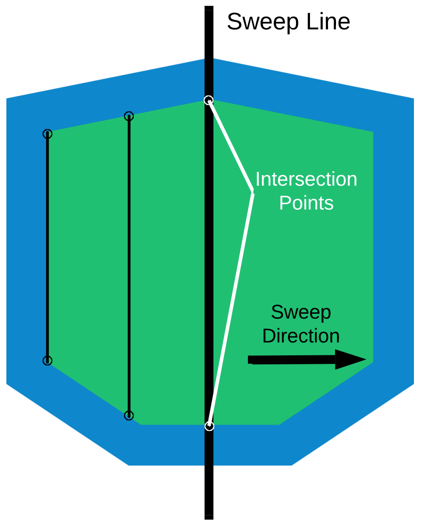


Figure 3.8: Line Sweep Layout Approach

The choice of configuration space and layout approach can have profound implications on the effectiveness and efficiency of the resulting constraints. In figure 3.9 we present the result of the line sweep approach using three different choices of configuration space: the vehicle configuration space, the sensor configuration space, and an intermediate option based on the smallest internal angle of the polygon defining the target area. As shown, the coverage rate for the sweep line method is best when the vehicle configuration space is used, because this allows the vehicle to get as close as possible to the target area boundaries. However, this comes at the expense of generally longer path constraints and potentially more transects required to fully cover the region. By choosing a configuration space with a larger offset, total length and number of constraints can be reduced at the expense of a slightly worse coverage rate. It is worth noting that in the case the payload footprint is smaller than the vehicle footprint, it is generally not feasible to use the sensor configuration space unless the vehicle is allowed to travel beyond the specified target area.

Beyond the conventional boustrophedon or spiral coverage patterns, the layout component can leverage additional domain data to distribute path constraints; for example, when the domain is defined over a region of moving water and the flow is known, suitable constraints may lie along the streamlines of this flow. Regardless of the implementation, given a problem description, this component should produce a list of path constraints that will ultimately be combined into a coverage path for the region.

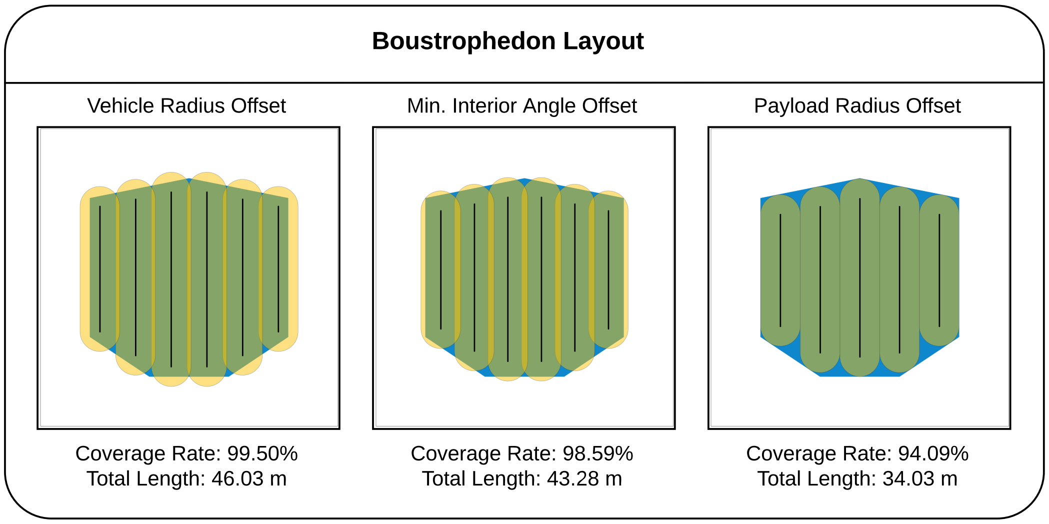


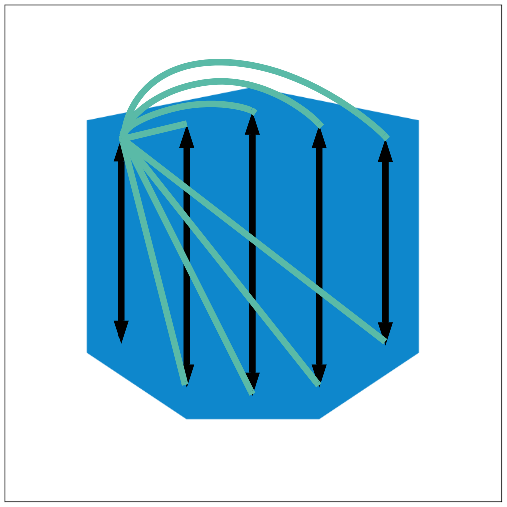
Figure 3.9: Examples of Boustrophedon Layout with Different Choices of Offset

3.2.2 Refinement

The refinement phase of the CB-CPP framework is unique in that it is the only optional stage; no refinement component implementation is necessary as long as the constraints created during layout are sufficiently descriptive to sequence or the downstream planner components can handle and resolve constraint ambiguities. If a refinement component is implemented, it will take the specified set of path constraints and use domain or user-specified knowledge to restrict additional parameters such as direction, thrust, or velocity. Although this will typically entail adding to the set of parameter constraints C, a refinement can also modify a path constraint's coordinate sequence X by inserting additional points along its defining geometry in order to increase the resolution of the representation and allow for different parameter constraints along any section of the path constraint.

The purpose of the refinement stage of the CB-CPP pipeline is twofold: the additional information encoded in the path constraints in this phase can be used directly by the vehicle during execution of the final coverage path and/or it can be leveraged at further points in the planning process in order to influence the assembly of path constraints into a coverage path. For example, we consider a refinement that makes use of a known communications envelope that denotes an area within the target coverage region in order to mark sections of each path constraint where reliable communications are unlikely. A robot executing a coverage path built from these constraints can then strategically throttle or disable its communications radio along segments of the path in order to conserve energy. Furthermore, this information can be used in subsequent planning steps to govern the order or direction in which path constraints are assembled, resulting in a final coverage path the minimized the time the vehicle spends between successive communications, perhaps by staggering the execution of path constraints that lie predominantly out of communications range.

The output of a valid refinement component will be another list of constraints, making it possible to chain multiple refinement components together during this stage, provided that they do not contradict one another. This design feature not only allows for the decomposition


(a) Connectivity of Unrefined Constraints

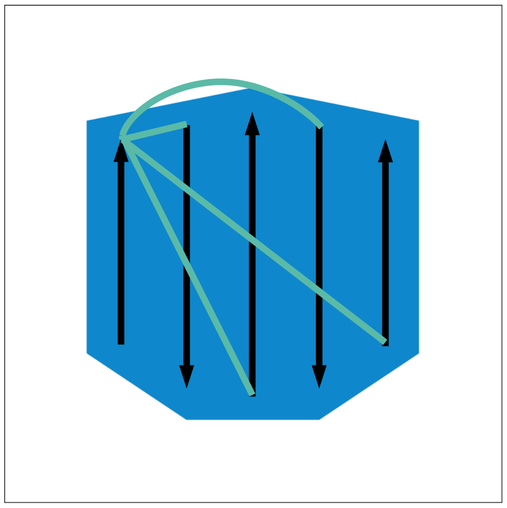

(b) Connectivity of Refined Constraints

Figure 3.10: Constraint Refinements can Reduce Search Space During Sequencing

of complex path constraint refinement into simpler distinct modules but also facilitates the modification of existing coverage planner implementations within the framework to add new optimizations or influence the behavior of the vehicle during execution.

3.2.3 Sequencing

After the set of path constraints is fully refined, the planning process proceeds to constraint sequencing. The sequencing component takes an unordered list of constraints and returns an ordered list that defines the order in which the constraints should be executed in the final coverage path. In addition, this component can take into account the ingress and egress points to the target coverage area, if specified. This can be useful if using a CB-CPP planner in conjunction with a cellular decomposition technique, in order to optimize the coverage pattern within each cell according to the determined tour through the cell adjacency graph.

Although sequencing components should generally be implemented to handle ambiguity in path constraints that have not been refined, the amount of refinement in the previous planning phase can have a significant impact on the computational load in this step. If the ingress and egress points of constraints are not fully defined, the sequencing component will have to consider all possible combinations of valid ingress and egress points during planning and fully constrain these parameters before returning the ordered list. Simply defining a direction of travel along each constraint, however, can reduce the search space during sequencing by half. An example illustrating this effect with a set of sample constraints is provided in figure 3.10, where the connections that must be considered during sequencing for a single constraint endpoint are shown in green. This effect is magnified when planning using closed constraints that each may have more than two possible ingress and egress points.

The output of a valid sequencing component should be an ordered list of path constraints that each have a single ingress and egress point defined, which can easily be assembled into a coverage path in the following stage.

3.2.4 Linking

The linking stage is the final phase of the constraint-based coverage planning pipeline where the ordered list of path constraints is combined to create the final coverage path for the target region. The simplest implementation of a linking component would be to take the ordered list of constraints and connect the egress of each constraint to the ingress of the following constraint with a single line segment, although depending on the target region geometry, this straight line segment may cross a region or obstacle boundary. A more robust and configurable approach is to design a linking component based around a point-to-point planner such as A*. Instead of simply connecting egress points to subsequent ingress points, this planner can now be used to determine collision-free optimized paths between constraints. More advanced point-to-point planners that take advantage of additional domain knowledge to improve path characteristics such as energy-efficiency or travel time can also be implemented for this component, at the cost of some additional complexity and computational load.

3.2.5 Summary

We have described the function of each stage in the constraint-based coverage planning pipeline shown in figure 3.5 and have hinted at how the conventional back-and-forth boustrophedonic or spiral patterns can be implemented within this framework. In the next section we walk through the design and implementation of the boustrophedon and spiral coverage patterns using the constraint-based approach and propose an alternative streamline boustrophedon approach that is well-suited for coverage of domains containing moving fluids such as rivers.

3.3 Case Studies

In this section we walk through a few sample implementations of traditional coverage planners within constraint-based framework, highlighting the reusability of planner components across implementations. In addition, we introduce a new streamline boustrophedon planner that has several convenient features for planning coverage paths in sections of river and only requires the development of a new layout component. For clarity and ease of explanation, the vehicle and payload footprints for these studies are fixed as congruent circles with a one meter diameter, while the target domains are restricted to simple scenarios: a 10 meter by 10 meter square for the first two studies and a 90 degree river bend with a uniform 10 meter width for the final study.

3.3.1 Boustrophedon Pattern

In our first case study, we describe one possible implementation of the popular boustrophedon coverage pattern as a constraint-based planner. Although there are different methods to build such a planner within the constraint-based framework, we present an approach with a single component for each stage of the CB-CPP pipeline, and generalized implementations that can easily be mixed and assembled into different coverage planners. The intermediate and final outputs of our planner on a simple coverage planning problem are shown in figure 3.11.

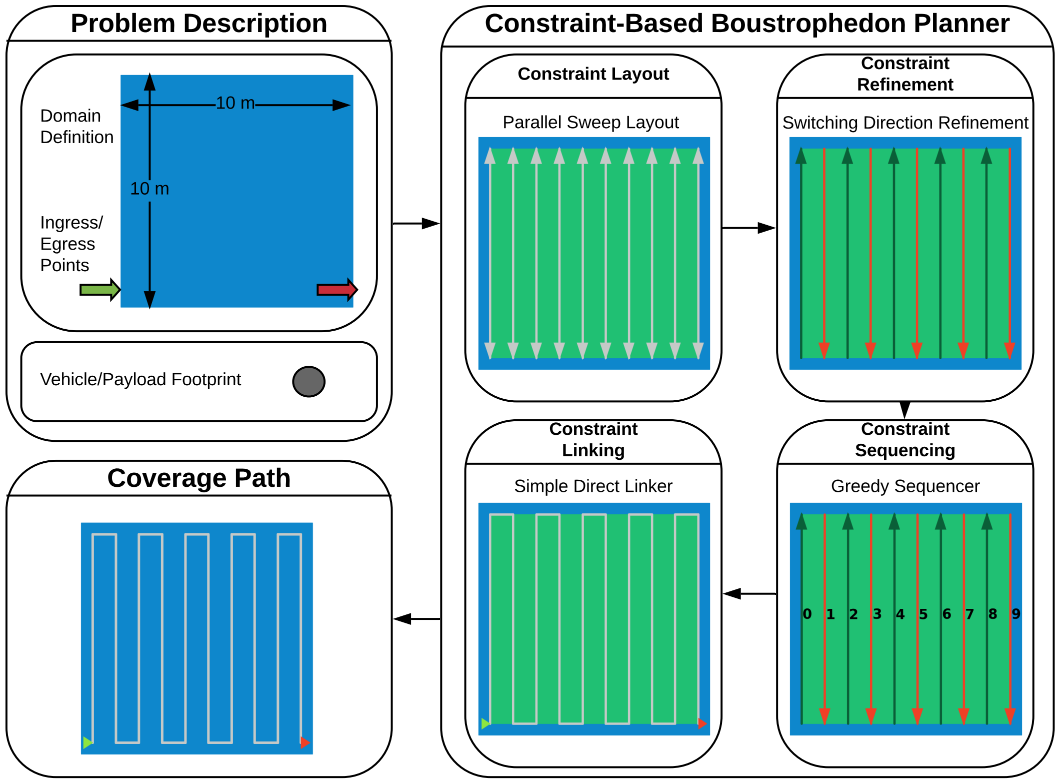


Figure 3.11: Sample implementation of boustrophedon coverage within CB-CPP framework

The layout component of our implementation is responsible for determining the initial spatial path constraints that will be used throughout the planning process. Given the 10 by 10 meter domain as a target area with congruent vehicle and payload footprints defined by a circle with a radius of 50 cm, our layout component begins by computing the configuration space of the vehicle within this target area, pictured as a green square within the blue domain in figure 3.11. Next the number of passes required to cover this configuration space in the desired direction is computed using the payload footprint; in our example we orient our passes vertically, and therefore since the width of the configuration space is 9 meters and our payload footprint radius is 50 cm, we can completely cover this area with 10 passes. The distance between parallel passes is chosen such that the first and last passes lie on the edges of the configuration space and the perpendicular distance between neighboring passes is no greater than two times the payload footprint radius. Although this works out to 10 passes with no overlap in our example, other geometries may require some overlap of payload footprints along parallel passes.

In order to generate the spatial path constraints corresponding to these passes, we sweep a line through the configuration space perpendicularly to the desired pass orientation, and record the intersection points of this line with the configuration space polygon. We begin at the first intersection of this line with the configuration space, which may be at a single point, and move the computed distance between passes repeatedly until the sweep line no long intersects the configuration space. Each set of intersection points recorded along the way becomes the coordinate list for an undirected path constraint. The output of this component for our example problem is shown in the constraint layout block in figure 3.11.

The next component in our implementation is constraint refinement, where each constraint is assigned a direction in an alternating fashion, such that any two adjacent constraints are oriented in opposing directions. Additionally, the orientations of the closest constraints to the ingress and egress points are chosen to minimize the distance from the ingress point to the beginning of the first constraint as well as the distance from the end of the last constraint to the domain egress point. The oriented constraints produced by this component for our example problem are pictured in the constraint refinement block in figure 3.11.

After the path constraints have been refined with an execution direction, they are passed to the constraint sequencing component of our planner, which determines an ordering that will be used to assemble the constraints into a complete coverage path. Our sequencer implementation uses a greedy approach, which relies on repeatedly searching for the closest ingress point across all remaining unsequenced constraints. The distance to each ingress point is measured with respect to a reference point, which initially starts at the domain ingress point specified by the problem and is updated to the egress point of each constraint as it is sequenced.

During its search, the greedy sequencer must consider all possible ingress points for each unsequenced constraint. If the direction of any constraint is undefined, the greedy sequencer resolves this ambiguity during operation at the time the constraint's ingress points was found to be closest to the current reference point. For open path constraints, specifying a single ingress point will collapse its sets of ingress and egress points to singletons, thereby implicitly defining a direction of execution for the constraint. Although constraint sequencing components should be designed to handle completely unrefined constraints with ambiguous ingress and egress points, there are benefits to some refinement if it is possible; by refining the direction of our input constraints in the previous stage of the planner, each constraint is limited to a single ingress point, thereby reducing the sequencer search space by half. This effect was illustrated during our earlier discussion of the constraint refinement phase in figure 3.10. The sequenced constraints produced by this component for our example problem are shown in the constraint sequencing block of figure 3.11.

With an ordered list of oriented path constraints, the final component of our implementation links the egress points of each constraint to the ingress point of the following constraint, and combines the constraints into a final coverage path. Because our sample problem involves coverage of a convex region, our implementation simply links each egress point to the next ingress point directly with a straight line, which is guaranteed to lie within the configuration space. The final coverage path determined by our planner for the example problem is presented in the coverage path block of figure 3.11.

This description of our constraint-based boustrophedon planner demonstrates how one of the most commonly-used CPP methods can easily be implemented within our constraint-based framework and offers a preview to the advantages of this approach over typical ad hoc or one-off implementations - namely the reusability and extensibility of individual components that make up a constraint-based planner. We showcase these features in the following case studies.

3.3.2 Spiral Pattern

In our next case study, we examine how the popular spiral coverage pattern can be implemented within the constraint-based framework. To accomplish this task, we will reuse two of the exact planning components used in the previous case study, without any modifications: the greedy constraint sequencer and the simple direct constraint linker. To further demonstrate the capabilities of our framework, we also implement this coverage planner without any refinement component, allowing the sequencer to operate on an unpruned search space and resolve any constraint ambiguities on its own. The intermediate and final outputs of our spiral coverage planner on the same planning problem considered in the previous case study are shown in figure 3.12.

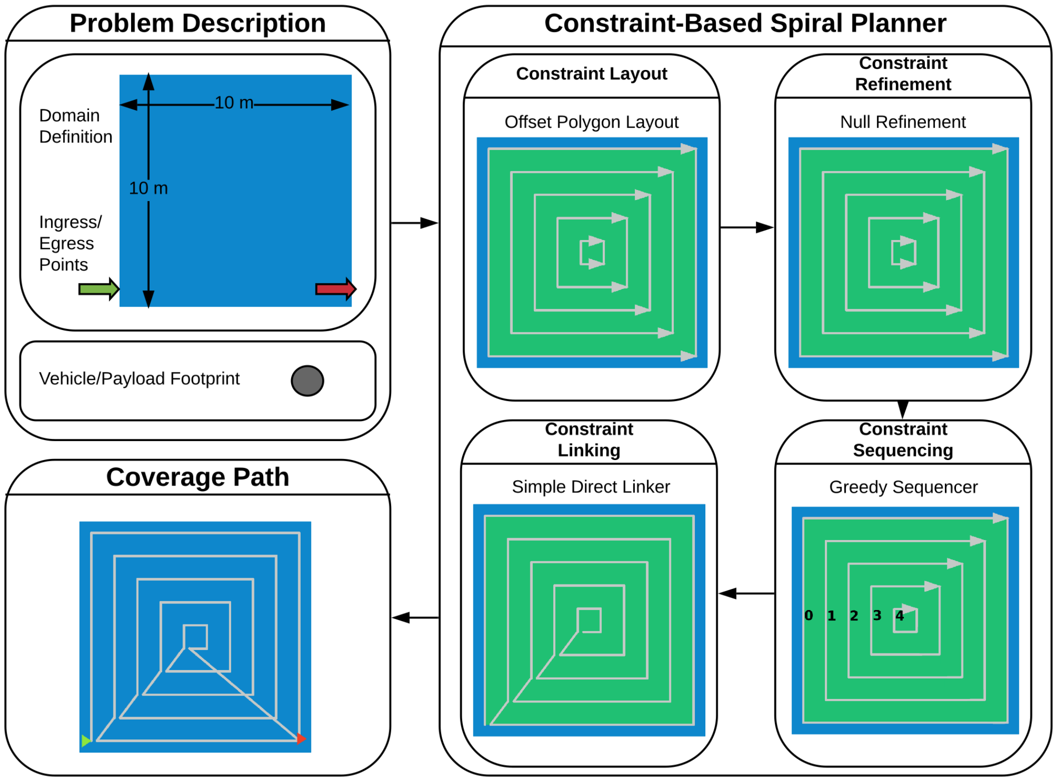


Figure 3.12: Sample implementation of spiral coverage within CB-CPP framework

The primary difference between the constraint-based boustrophedon planner presented in the previous case study and the constraint-based spiral planner is the manner in which the path constraints are laid out throughout the target coverage region. As before, the layout component begins by computing the configuration space according to the domain and vehicle footprint geometry, pictured as a green square in figure 3.12. Instead of sweeping a line through this configuration space, however, the spiral pattern layout module defines constraints by repeatedly offsetting the contour of the configuration space inwards. The specific process used to generate each successive offset contour can grow complicated depending on the complexity of the target region, vehicle and payload footprints, and the degree of coverage and overlap desired. The approach implemented for this case study translates each edge of the prior contour a fixed distance along its normal vector towards the center of the polygon and then uses the intersection points of these new edges to determine a set of one or more offset contours.

At each offset step, the newly determined contours are used to instantiate new path constraints that will be passed along to components further down the planning pipeline. In contrast to the open path constraints produced by the boustrophedon layout in the previous case study, the path constraints generated by this spiral layout implementation are closed path constraints, which can be joined together at one of any number of defined transition points along their defining curves. As mentioned in our introduction to path constraints, any closed constraint can also be transformed into a single open constraint with coincident endpoints, however, this will in effect restrict the transition point for each contour. Alternatively, any closed constraint can also be decomposed into a set of open constraints, although this will generally remove the restriction that the resulting constraints are executed in order. In this case study we want to follow a spiral coverage pattern where the vehicle will move along each contour completely before transitioning to another contour, therefore we do not perform this decomposition. The output of this component for our example problem is shown in the constraint layout block in figure 3.12.

The next stage in the CB-CPP pipeline is constraint refinement, which we have omitted in our implementation for illustration purposes. Consequently, the set of closed path constraints created during layout is passed directly into the sequencing module of our planner, which is the same greedy sequencer implemented for the boustrophedon planner in the previous case study. Unlike that example, however, the set of closed constraints supplied to the sequencer have undefined directions and transition points, which means that the full set of ingress points for each constraint must be considered during sequencing and a direction to execute each constraints must be determined. This ambiguity is resolved in a greedy fashion, where the sequencer repeatedly uses the shortest distance metric to select and fully define the next constraint.

The greedy sequencer initially starts at the domain ingress point specified by for the problem and considers all possible ingress points across all unsequenced constraints before choosing the closest option. This point is then selected as the ingress point for its parent constraint, which for closed path constraints corresponds to defining a transition point that subsequently collapses the sets of possible ingress and egress points to singletons containing that point. The parent constraint is then indexed and removed from the set of unsequenced constraints. This process is then repeated starting at every possible egress point of the prior constraint, and continues until no unsequenced constraints remain.

One distinction between open and closed path constraints is that while an open constraint's direction is fully defined when the sets of possible ingress and egress points are singletons, this information is insufficient to define the direction for a closed constraint. Since the ingress and egress points defined by our choice of transition point are coincident, however, the direction of travel along each constraint will not influence the greedy selection of the following constraint and will not affect the total length of the resulting coverage path. Due to these circumstances, our greedy sequencer implementation will assign a default clockwise execution direction if it determines that selecting an ingress point did not fully define the direction of a processed path constraint. In more complex sequencer implementations, the direction of a closed constraint may be defined using other methods such as minimizing the change of a vehicle's heading required when moving between constraints. The sequenced constraints produced by our implementation for the example problem are shown in the constraint sequencing block of figure 3.12.

Once an ordered list of fully defined path constraints has been prepared, it is passed to the final linking component of our coverage planner. Since the target region for this case study is convex without any holes, the simple direct linker module created for the previous study is still suitable and can be directly reused without modification. The final coverage path determined by our planner for the example problem is presented in the coverage path block of figure 3.12.

Over the course of this and the previous case studies, we have described possible implementations of the two most common approaches to area coverage: the direction-parallel boustrophedon pattern and a contour-parallel spiral pattern. Furthermore, we have illustrated how the CB-CPP framework facilitates code reuse through the segmentation of the planning process into phases that are commonly implemented across individual planners. In the following section we propose a new streamline boustrophedon layout for coverage in rivers, and walk through the process of building a constraint-based planner around this layout technique.

3.3.3 Streamline Boustrophedon Pattern

CB-CPP offers the opportunity to easily develop a variety of new coverage path planning algorithms by simply mixing existing implementations of planner components or combining them with newly designed modules; even introducing a single new component into a planner implemented within the CB-CPP framework can dramatically change the characteristics and performance of the coverage paths it produces. In this final case study, we exploit this capability to create a new streamline boustrophedon planner for river domains, which generates coverage paths that follow approximate streamlines in the river. In order to accomplish this, we design a new streamline layout technique and pair it with the switching direction refinement and greedy sequencing components developed in previous case studies, as well as an upgraded linking module based on smoothed A*. The intermediate and final outputs of our streamline boustrophedon planner on a simple quarter turn target region are shown in figure 3.13.

The crux of this new planner is the new streamline layout component that generates curves based on the geometry of the river domain that can approximate the streamlines of the flow in the river. Just like the two CB-CPP planners discussed in the previous case studies, this layout module begins by computing the configuration space according to the domain and vehicle footprint geometry, which is represented by the green areas shown in figure 3.13. Next, the layout algorithm approximates the medial axis or centerline of the river using the supplied domain geometry. For the simple scenario in the case study, this axis can be computed directly from the equations defining the river banks, however, a variety of algorithms exist to determine the medial axis for arbitrary geometries using Vornoi diagrams [47], Delaunay triangulation [63], the medial axis transformation (MAT) [78], and the edges of a Triangulated Irregular Network (TIN) [56].

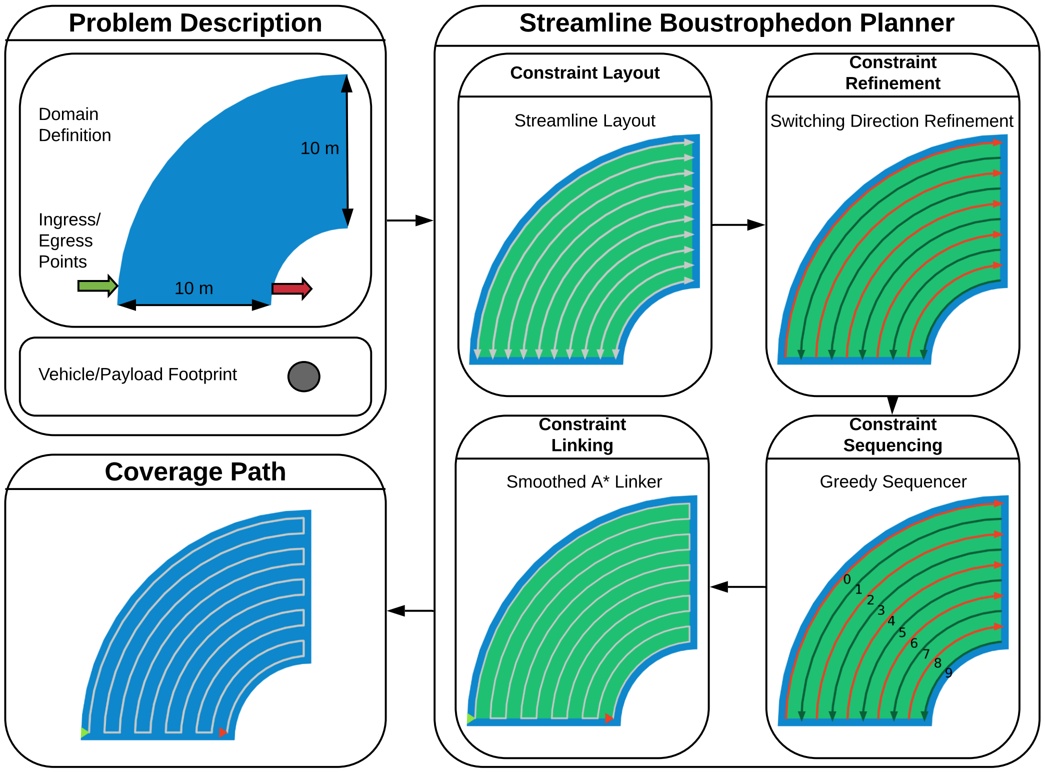


Figure 3.13: Sample implementation of streamline boustrophedon coverage within CB-CPP framework

After the central axis is determined, the layout process continues by examining perpendicular cross-sections of the river at regular intervals along this centerline. The length of the longest of these cross-sections, which corresponds to the width of the river at its widest point, is used to determine the number of transects N required to cover the entire target region given the vehicle payload footprint. The perpendicular cross-sections are then each divided into $N-1$ equal line segments, and their endpoints are numbered from left to right. These endpoints are then grouped together across all cross-sections according to their index to create the coordinate sequences that define the final path constraints. Additional details and explanatory figures for this constraint layout approach are available in our discussion of energy-efficient modifications to this layout component in chapter 4. The output of this component for our example problem is shown in the constraint layout block in figure 3.13.

The next two components in our planner implementation are reused directly from the constraint-based boustrophedon planner described in the first case study. The first of these is the switching direction refinement module, where each constraint is assigned a direction in an alternating fashion, such that any two adjacent constraints are oriented in opposing directions. Additionally, the orientations of the closest constraints to the ingress and egress points are chosen to minimize the distance from the ingress point to the beginning of the first constraint as well as the distance from the end of the last constraint to the domain egress point. After the path constraints have been refined with an execution direction, they are passed to the constraint sequencing component repurposed from our previous case studies, the greedy sequencer. As before, the greedy sequencer module repeatedly searches for the closest ingress point across all remaining unsequenced constraints. The distance to each ingress point is measured with respect to a reference point, which initially starts at the domain ingress point specified by the problem and is updated to the egress point of each constraint as it is sequenced. The oriented and sequenced constraints produced during these planning phases are pictured in the constraint refinement and constraint sequencing blocks in figure 3.13.

Once a sequenced list of fully defined path constraints has been prepared, it is passed to the final linking component of our coverage planner. In contrast to the two previous case studies, the target region in this scenario is concave, and therefore the previously used simple direct linker module is unsuitable because connecting the ends of two constraints with a straight line may intersect with the region boundary and lie partially outside the configuration space. For example, if the egress point of transect 9 were to be connected directly to the ingress point of transect 7 in the constraint sequencing block of figure 3.12, the direct connecting path would cut across the inner bank of the river. Although the constraint refinement we apply in our planner implementation may result in the simple direct linker producing a viable path for this specific scenario, implementing a dedicated point-to-point planner within a linking component is more appropriate for general river domains, which may meander significantly. In our linking component, we implement a post-smoothed A* planner to connect the ingress and egress points of the sequenced path constraints. The final coverage path determined by our planner in this case study is depicted in the coverage path block of figure 3.13.

3.3.4 Additional Scenarios

Although the case studies in this section were limited to simple scenarios and vehicle configurations, these example planners implemented within the CB-CPP framework can produce coverage paths for a variety of target region geometries as well as different combinations of vehicle and payload footprints. In order to showcase some of this versatility, the intermediate outputs and final coverage paths produced by these planners for additional scenarios are presented in appendix A.

3.4 Summary

In this chapter we formally introduced the path constraint abstraction and described a constraint-based coverage path planning (CB-CPP) framework based on assembling these path constraints into coverage paths through four distinct phases: Layout, Refinement, Sequencing, and Linking. After an examination of each of these planning stages, we presented a series of case studies illustrating how common coverage techniques such as the boustrophedon and spiral patterns could be implemented within the framework with significant code reuse. We concluded our case studies with the introduction of a new streamline boustrophedon planner, that only required the development of a new constraint layout planner component.

Throughout this chapter, we have targeted our discussion of the CB-CPP framework and subsequent examples of planners implemented within it in order to highlight several of the advantages over conventional approaches to coverage planner implementation. The most impactful of these are revisited and briefly discussed below.

- **Modular**
  By partitioning the planning pipeline into four distinct phases with well-defined inputs and outputs at each stage, development of coverage planner components can occur independently and concurrently. This can speed planner development and simplify the design of new optimizations that can be experimented with in isolation while holding other planner components constant. The four stages in the pipeline (layout, refinement, sequencing, and linking) additionally encourage a natural separation of steps in the planning process, without restricting developers from designing more complex effects based on interaction between components across multiple planning phases.

- **Reusable**
  Since many coverage planning algorithms include common subtasks or boilerplate that is not specific to the core approach but nevertheless required, the modular nature of planners implemented within CB-CPP allows for a tremendous amount of code reuse. As long as planner components are developed to adhere to the specifications defined by the CB-CPP framework, component implementations can often be directly dropped in when building new constraint-based planners.

- **Scalable**
  Due to the modular approach of the CB-CPP framework, the choice of component implementation at each stage in the pipeline can scale according to the available computational or time resources and adjust based upon any domain or application specific knowledge. For example, while a greedy constraint sequencing approach may be appropriate when planning with limited resources, a more computationally demanding heuristic sequencer may produce superior results. Similarly, if the target coverage area is known to be convex and free of obstacles and holes, a simple direct linking component will substantially reduce computational and memory requirements over a post-smoothed A* linking algorithm running on a dense grid map.

- **Extensible**
  Existing coverage planners within the framework can easily be augmented to imposed additional vehicle constraints or introduce additional optimizations, simply by adding new constraint refinement modules. For example, in order to constrain vehicle velocity within a specified ″safe″ subregion of the domain, a new refinement can be implemented that encodes a velocity limit as an additional constrained parameters along sections of each path constraint that lie within the specified subregion.

Thanks to its modular and extensible design, the CB-CPP framework can be used to build coverage path planners for a variety of domains and applications. Planning for multi-agent coverage can also be achieved through the implementation of suitable sequencing and linking modules that assemble multiple coverage paths instead of a single plan for the entire target region. Due to some reliance on domain knowledge at each phase within the CB-CPP planning pipeline, however, this framework is primarily suited to building offline coverage planners. More complex implementations that merge the planning and execution processes to achieve online coverage planning using the cb-cpp framework may be possible but are outside the scope of this work. Instead, this work focuses on developing offline coverage planning algorithms that are robust to some degree of uncertainty in domain knowledge.

The development of the CB-CPP framework was motivated by the objective of creating an energy-efficient coverage planner for river domains. To this end, path constraints and CB-CPP were designed to provide a clear abstraction for defining a vehicle's motion through a region and a methodology for interacting with this abstraction to plan efficient coverage by the vehicle. The phases of the CB-CPP pipeline allow for planner optimization at every stage, which lets users focus on a single process when trying to improve a planner. This was instrumental for the development of the energy-efficient coverage planners described in this work. In the next chapter we examine the details of the energy-efficient optimizations that power these planners.

Chapter 4

Energy-Efficiency Coverage Planning

In this chapter we address the fundamental problem that motivates this work: how to plan energy-efficient coverage paths in river domains, where currents in the fluid have substantial impact on the vehicle's motion. Although tackling this question as a single irreducible problem is daunting, the constraint-based planning architecture developed in the previous chapter provides a tool to decompose this problem into discrete stages that can then be optimized independently or in sensible combinations. This approach allows us to easily encode common sense strategies based on available domain knowledge by building new specialized components within the CB-CPP framework, and reuse standard existing modules when appropriate.

In this work we propose three avenues for the optimization of coverage path plans to reduce the energy-usage during execution: Streamline Constraints, Constraint Coverage Overlap Biasing, and Flow-Aware Constraint Refinement and Sequencing. These optimizations span different stages of the CB-CPP framework and can be applied independently or sequentially if appropriate for the application scenario. In the following sections, we delve into the specifics of each approach and describe the reasoning behind each method.

4.1 Streamline Constraints

Our first optimization approach centers around the streamline constraints introduced during the development of the streamline boustrophedon planner in the final case study of chapter 3. The idea behind these constraints is that their defining geometries approximate the streamlines of the river, a feature which can simplify the coverage path planning process and open up possibilities for further optimizations. In this section we will discuss the motivation and potential benefits of streamline path constraints, before walking through our proposed layout methodology that can be used to generate these constraints for a given target region of river.

4.1.1 Motivation

Common geographical features of rivers, such as meanders, bifurcations, confluences, and river islands, can make for complex geometries when defining a target region for coverage. While conventional approaches to coverage path planning may handle this complexity by decomposing the target region into simpler cells before planning a coverage path for each cell individually, streamline constraints offer a convenient alternative. By following approximations of the river's streamlines that naturally curve with the riverbanks and around obstacles, streamline constraints can be distributed to cover even complex sections of rivers without the need for explicit cellular decomposition prior to planning. Furthermore, because the endpoints of streamline constraints tend to be clustered together at upstream and downstream points throughout the target region of river, constraint linking is simplified and the resulting coverage paths can end up with significantly less backtracking and turns than would result from cellular decomposition and direction-parallel boustrophedon coverage.

Beyond the aforementioned geometric advantages, streamline constraints additionally facilitate further energy saving optimizations based on the currents of the river. Because the direction of flow in the river will generally correspond to the direction of the approximated streamlines that define the constraints, this greatly simplifies reasoning about the additional energy costs of executing a streamline constraint due to the river currents. For example, if we consider a linear path constraint defined by the cross-section of the river and a single streamline constraint, it is clear that the energy costs of traversing the streamline constraint in the upstream direction will be higher than a downstream traversal, however, the same cannot be easily determined for the cross-section constraint without additional information regarding the river current distribution.

In addition, the alignment of streamline constraints with the surface currents in the river also makes them well suited to additional refinement that can be used to enforce more efficient execution by a specific vehicle. Perhaps the clearest example of such a refinement is to limit the thrust of the vehicle along streamline constraints when travelling downstream. Since thrust makes up such a large component of a vehicle's energy consumption, limiting thrust output to just allow for small course corrections while drifting with the currents along a constraint can result in significant energy savings. Depending on the capabilities of the vehicle, streamline constraints open the door for many other optimizations such as adjusting ballast or deploying a drogue to adjust the impact of river currents on its motion.

4.1.2 Layout Methodology

Like many of the other planner components described during our discussion of the CB-CPP framework, the specific implementation of a layout module that generates streamline constraints can be accomplished in a variety of ways. Although the computation of true streamlines requires knowledge of the complete vector field representing the surface currents throughout the target region, this information is seldom available ahead of time and generally requires taking measurements in the field. While it is perfectly feasible to design a CB-CPP algorithm with a layout component that requires a surface current vector field input alongside the target area, in this work we chose to approximate the streamlines using the medial axis of the river instead. The resulting streamline approximations are sufficient to implement the simple optimizations we discussed above and remove the need for a prior survey of river currents before our planner can develop a coverage plan.

In the discussion of our streamline constraint layout methodology that follows, we refer to an example scenario involving a section of the Monongahela River in Pittsburgh, which is presented in figure 4.1. In this scenario, streamline constraints are generated and assembled into a boustrophedonic coverage path by the streamline boustrophedon planner introduced in chapter 3, however, since the streamline constraints generated by our layout module are valid path constraints, they can be used within any other CB-CPP planner as well.

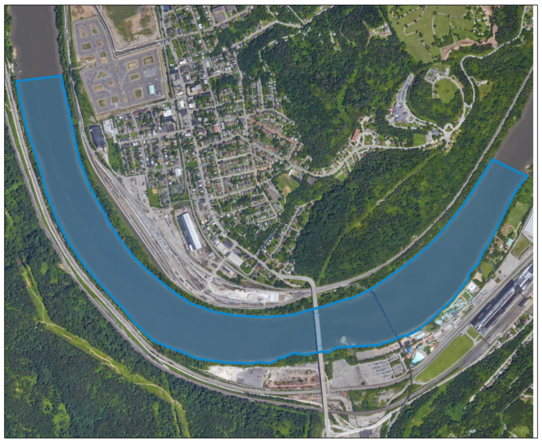
The inputs to our streamline layout module are the vehicle and payload configurations, target coverage region, and coordinates of relevant river geometry in the section of river surrounding the target region. This is a slight departure from the minimal inputs into typical constraint layout algorithms in that the user is also asked to specify the coordinates for river geometry that would impact its flow, even if those features lie outside the specified target region. In our example scenario, these features are simply the banks of the river, but for more complex stretches of river they may include the boundaries of islets within the river or artificial structures such as piers or bridge pylons. This additional geometry is used by our layout algorithm to approximate the centerline or medial axis of the river, the first step towards generating approximate streamlines.

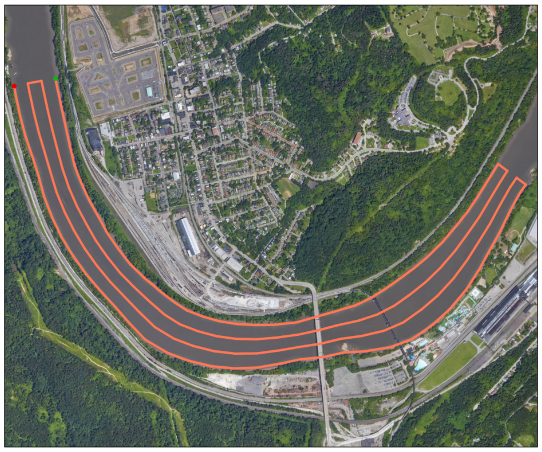
When the river geometry is simple enough to be described exactly by a set of equations, the medial axis can be computed directly, however, this is seldom the case in real world scenarios with natural rivers. In order to implement a general approach to laying out streamline constraints, we must make use of an algorithm capable of determining the medial axis of arbitrary geometries. As we hinted at in the final case study in the previous chapter, there are many existing algorithms that solve this problem including techniques that leverage Delaunay triangulation [63], the medial axis transformation (MAT) [78], and the edges of a Triangulated Irregular Network (TIN) [56]. However, perhaps the most popular approach adopted many medial axis algorithms relies on building a Voronoi diagram from the input geometry [19,47,62,80].

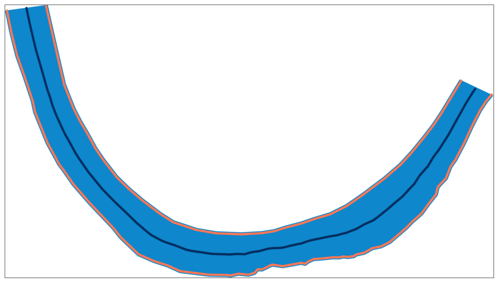
In our implementation, we determine the river centerline using a methodology adapted from several existing algorithms that use the Voronoi diagram. The availability of Voronoi diagram functionality within the SciPy library played a large role in our decision, as this simplified our implementation and provided well-tested functions capable of processing complex geometries. The first step in our approach is to pre-process all the provided river geometry in order to convert each curve into a dense set of points. These points are then used to construct the Voronoi diagram for the scenario.

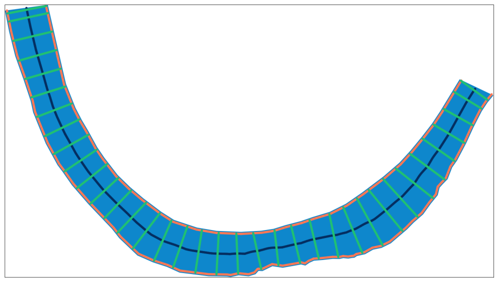
Once the Voronoi diagram has been created, we extract from it all the ridge edges that do not intersect the original river geometry and assemble them into a graph. At this stage in the process, the graph is a skeleton of the river geometry and contains an approximate centerline of the river. In order to extract this centerline, we first select all the nodes with degree 1 within the graph. These nodes represent all the endpoints of the river's skeleton and therefore a path between some pair of them will approximate the medial axis of the river. For simple river scenarios without bifurcations, it suffices to determine the shortest path between all combinations of endpoints and select the path with greatest length as the centerline. The river bank geometry inputs and resulting centerline computed for our example scenario are shown in orange and black respectively in figure 4.1c.

After the medial axis of the river has been determined, our layout module proceeds to compute perpendicular cross-sections of the river at regular intervals along its centerline. These cross-sections extend from the centerline to the boundaries of the target coverage region, which may or may not correspond to actual river geometry. In the case where


(a) Target Region on the Monongahela River


(b) Streamline Boustrophedon Coverage Path


(c) River Centerline Computation


(d) River Cross-Section Analysis

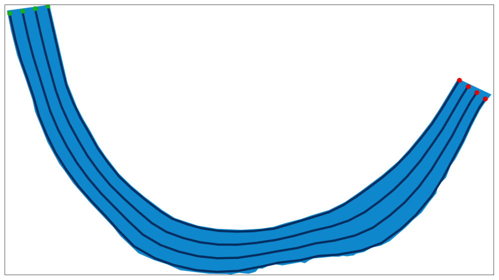

(e) Streamline Constraint Layout

Figure 4.1: Monongahela River Bend Example Scenario

the target region encompasses the entire river and the river banks are coincident with the boundary of the target region, the computed cross-sections are true cross-sections of the river, but this is not a necessary condition. A sample of these cross-sections for our example scenario is shown in figure 4.1d, where each cross-section is denoted in green.

After generating these cross-sectional segments, our proposed layout algorithm then considers the lengths of these perpendicular segments in order to determine the maximum width of the target region. In the case that the target coverage region spans the entire river, this value corresponds to the width of the river at its widest point. The algorithm then uses this value along with the vehicle payload footprint in order to determine how many streamline constraints are needed to cover the entire target region. If it is determined that N constraints are required, each perpendicular cross-section is then divided into $N - 1$ equal line segments, and their endpoints are numbered from left to right with respect to the river's centerline. These endpoints are then grouped together across all cross-sections according to their index to create the coordinate sequences that define the final path constraints. The streamline path constraints for our example scenario are presented in figure 4.1e, while the final coverage path produced from these constraints by the streamline boustrophedon planner is shown in figure 4.1b.

Although this discussion has described our implementation of the streamline constraint layout methodology in the context of a simple section of river without bifurcations or obstacles, one of the claimed benefits of streamline constraints is that they should allow for coverage of more complex river geometries without the need for prior decomposition. We address this point in the following subsection where we describe how our layout algorithm can handle junctions along a river and present an example scenario centered around the confluence of the Allegheny, Monongahela, and Ohio Rivers near Pittsburgh, PA.

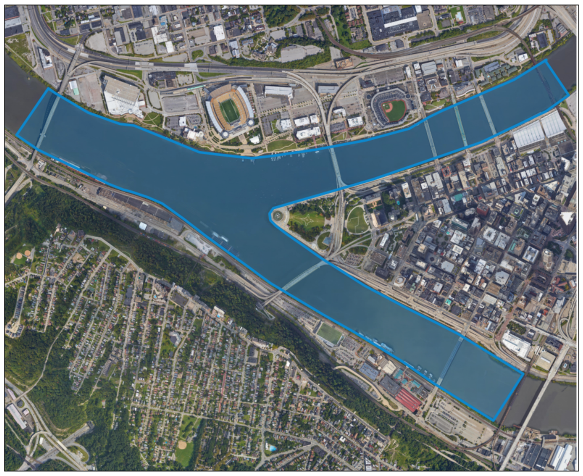
4.1.3 River Junction Extension

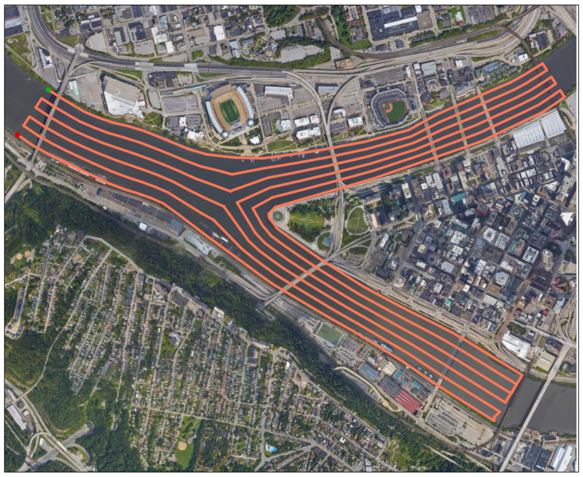
Apart from an adjustment to our medial axis approximation algorithm, the methodology we have proposed for laying out streamline constraints will largely function for river sections containing any junctions, whether they be due to tributaries flowing into the river or the channel splitting upstream of a river island and coming together again at a point downstream. Due to the larger cross-sectional widths that can occur at junction points, however, the resulting streamline constraints can end up distorted around the junction points and may also be distributed more densely than required for complete coverage. In order to address these issues, we have developed an extension to our layout methodology that uses alternatively defined cross-section segments near river junction points. In the discussion that follows, we refer to an example scenario involving the confluence of three rivers near Pittsburgh, which is presented in figure 4.2.

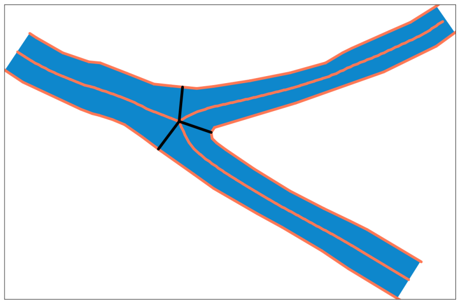
As before, the first stage of our extended layout algorithm involves approximating the centerline of the river using our Voronoi-based approach. After pre-processing the river geometry, generating the Voronoi diagram, and building a graph from the diagram's ridge edges, we deviate from our previous approach and execute a search for junction points in the graph instead. These points will be represented by nodes with a degree of 3 in the graph, however, this relationship is not a bijection and there may be additional nodes with degree 3 that do not represent junction points, and are merely short branches off the centerline that occur as a consequence of the discretized river geometry.

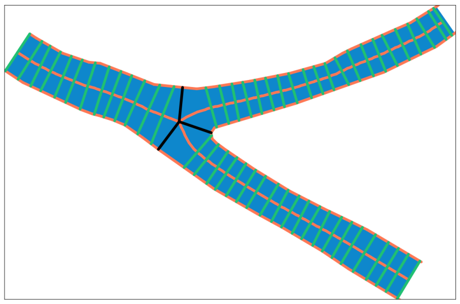
In our previous centerline algorithm we were able to filter out these extraneous branches by simply finding the longest shortest path between two endpoints in the graph, but applying this technique in this case would filter out true junction points as well. Fortunately, the length of extraneous branches off the centerline is limited to approximately half the river width at any given point, which is typically substantially shorter than the stretch of river on either side of a true junction point. Therefore, in most scenarios, we can use the length of the branches on either side of a junction point candidate to determine whether it is a true junction point or an extraneous branch off of the centerline.

After we have identified the true junction points within the ridge edge graph, we can split this graph up at the nodes corresponding to each of these points and once again determine the longest shortest path between endpoints for each of of the new sub-graphs. These paths will correspond to the centerlines for the specific sections of river represented by each


(a) Target Region at the Confluence of Three Rivers


(b) Streamline Boustrophedon Coverage Path


(c) Centerline Computation and Partitioning


(d) River Cross-Section Analysis

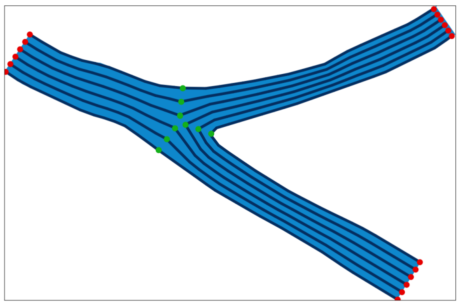

(e) Streamline Constraint Layout

Figure 4.2: Three Rivers Confluence Example Scenario

sub-graph. The centerlines determined using this extended methodology for our example scenario is shown in figure 4.2c.

By splitting up the ridge edge graph at the junction points, we essentially partition the target region into simple sections of river which can each be processed by our layout algorithm independently. As we alluded to earlier, however, this can result in distorted cross-sections near junction points and excessively dense constraint distribution. To resolve these issues, we propose a final extension to our layout module, which computes a special cross-section for each centerline that meets at a junction point. In order to construct these cross-sections, we simply draw a line from each junction point to the closest point on each disjoint curve defined by the river geometry, and select the three shortest lines. For each centerline, the special cross-section is defined as the polygonal chain comprised of the two lines at the junction point that have the smallest angles between them and the centerline. The lines that make up these special cross-sections for our example scenario are shown in black in figure 4.2c.

After the special cross-sections have been constructed for all centerlines and junction points, the constraint layout algorithm can proceed with the computation of perpendicular cross-sections at regular intervals along each centerline, with the caveat that these perpendicular segments cannot intersect any special cross-sections. A sample of valid cross-sections determined for our example scenario is shown in figure 4.2d, where the perpendicular and special cross-sections are shown in green and black respectively. The remainder of the layout process plays out as before, with each cross-section being split up into line segments of equal length according to the vehicle payload footprint and the length of the longest cross-section. The endpoints are then grouped together across the sets of cross-sections corresponding to each centerline in order to generate streamline path constraints for each simple river section. The computed constraints for our example scenario are shown in figure 4.2e, while the final coverage path produced from these constraints by the streamline boustrophedon planner is shown in figure 4.2b.

With the changes proposed in this subsection, we have extended the capability of the streamline constraint layout algorithm to process sections of river with junction points. This opens up the possibility of using this layout module during planning for more complex scenarios involving river islands and multiple bifurcations, which can be represented as simple stretches of river joined together with junction points. Coverage plans developed using this layout method for additional real world scenarios are presented in table A.5 in appendix A.

4.1.4 Summary

In this section we introduced the concept of streamline constraints - a specialized building block of coverage path plans that uses the streamlines of the ambient current field as their defining geometry. We discussed the advantages of using streamline constraints during coverage planning in real world river domains and the promising opportunities for further optimizations that these constraints provide. Finally, we presented our implementation of a constraint layout algorithm that can generate streamline constraints based on an approximation of a river's streamlines from its geometry, without a priori knowledge of the surface current field.

Although streamline constraints appear to be well-suited technique for coverage planning in river domains, they are not without shortcomings, particularly when applied to long stretches of the diverse and complex river geometries found in nature. Since the cross-sectional width of rivers typically varies along its length, the likelihood of significant coverage overlap increases with the length of the target region of river being planned on. Fortunately, this increase in coverage overlap affords some additional latitude in the arrangement of the constraints on the river. In the next section we propose another optimization that exploits the flexibility provided by coverage overlap to reduce the execution costs of the final coverage path.

4.2 Constraint Coverage Overlap Biasing

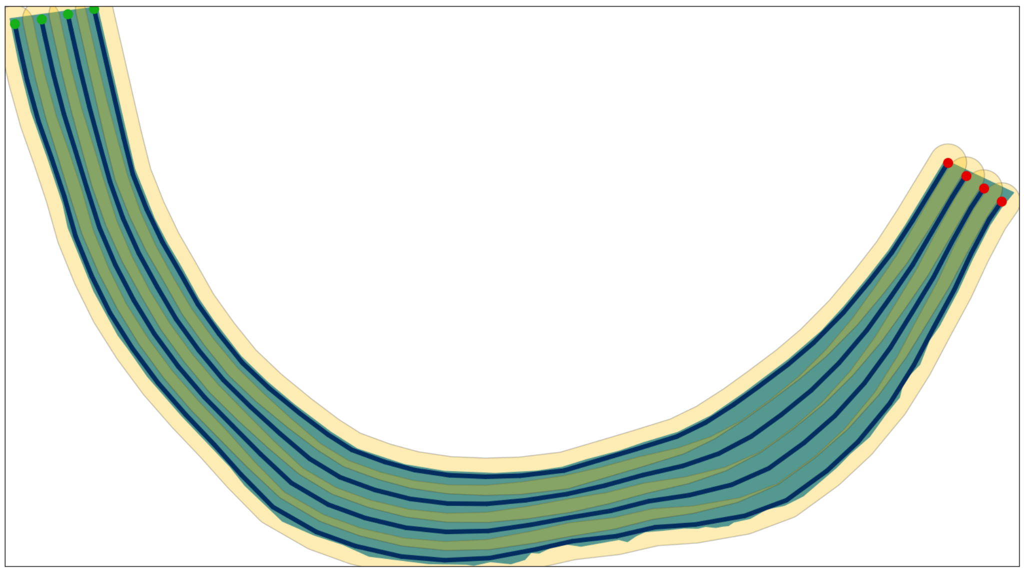
Although one of the goals of coverage path planning is to minimize redundant coverage during execution, some degree of coverage overlap is typically unavoidable. While there has been work done in the past to develop techniques that plan ahead in order to ensure that the area lying along a vehicle's return path is not visited repeatedly [20], there has not been much consideration for the inherent coverage overlap that occurs when the dimensions of the target region are not perfectly divisible by the payload radius of the vehicle. When the individual transects that make up a coverage path are simple straight lines, as is the case with the prototypical implementation of the common boustrophedon pattern, and there is no difference in the cost of executing a transect regardless of its location within the target region, the distribution of coverage overlap throughout the survey area may have little bearing on execution costs. However, as soon as the straight transects are replaced with more complex curves and heterogeneous environmental conditions throughout the target region are considered, the consequences of where redundant coverage occurs can have a significant impact on the performance of a coverage plan.

Our second optimization approach centers around the idea of modifying the layout of path constraints throughout the target region in order to decrease their execution costs while still ensuring complete coverage through the exploitation of coverage overlap. Fundamentally, our proposed method biases the distribution of path constraints towards more advantageous sectors within the survey area, resulting in a sparser allocation of constraints across problematic regions. In the context of river domains that are the focus of this work, our approach is to bias coverage towards river geometry with a higher curvature, such as the inner bank of a meander.

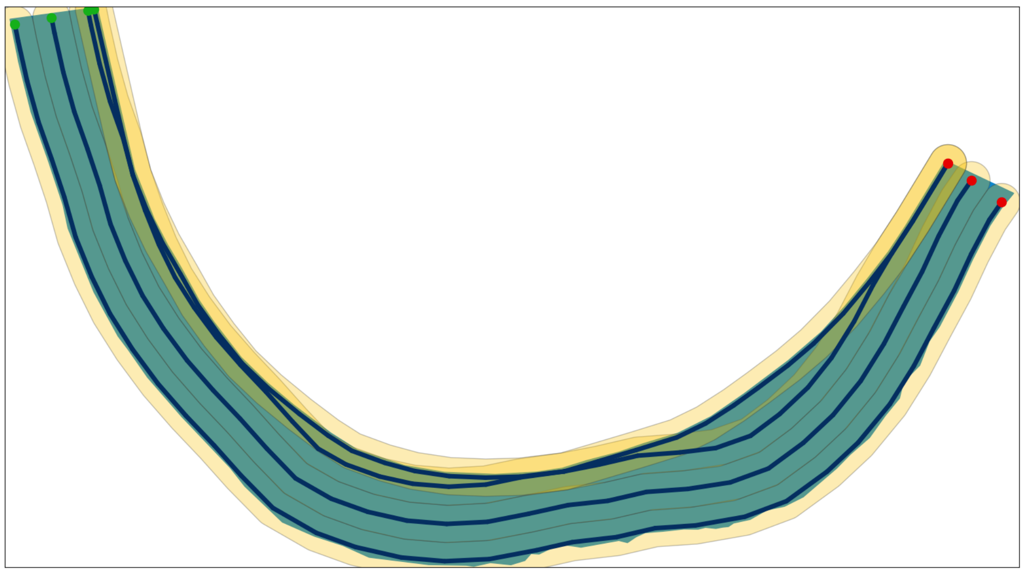
When planning coverage of sections of river that include bends, the advantages of paths that lie closer to inner bank are twofold. As flowing water is forced to change direction with the river, inertial forces push the faster-moving volume from the center of the river channel towards the outer bank, which creates an asymmetric current distribution with slower flow near the inner bank [55]. Secondly, because the curvature of the inner bank is higher than the outer bank, the streamlines of the river's current field are generally shorter at the inner bank than at the outer bank. Our proposed optimization seeks to leverage these properties during the layout of streamline constraints in order to reduce the sum of the generated constraint lengths and provide more routes through slow-flow regions that a vehicle can exploit to move upstream.

The implementation of this optimization only requires a simple modification to the streamline constraint layout procedure discussed in the previous section; when partitioning the cross-sections that have been computed along the river's centerline into line segments, we maximize the length of segments closest to the river geometry with lowest curvature. In effect, this will increase the spacing of streamline constraints that lie closer to the bounding river geometry with lower curvature, consequently increasing the density of constraints in higher curvature regions. In order to further explain this idea, we consider the streamline constraints generated for the Monongahela River scenario discussed in the previous section. The original uniformly distributed constraints for this scenario are presented alongside the biased streamline constraints in figure 4.3, with the coverage envelope associated with the vehicle payload footprint overlaid on each transect in transparent yellow.

An examination of the constraint coverage envelopes and their darker overlapping regions in figure 4.3 reveals how the choice of where coverage overlap occurs within the target region



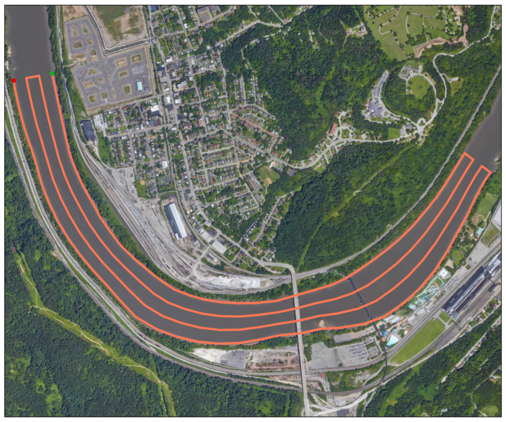
(a) Coverage of Uniformly Distributed Constraints



(b) Coverage of Constraints Biased Towards Higher Curvature Bank

Figure 4.3: Biasing the Coverage Overlap of Streamline Constraints

can result in substantially different plans. When path constraints are laid out uniformly, coverage overlap is likewise distributed more uniformly throughout the target area. However, when path constraints are biased towards the inner bank of the river, the coverage overlap is clustered near this bank as well. In this case, the path constraints near the outer bank of the river are spaced as far apart as the vehicle payload footprint allows, eliminating coverage overlap in this zone altogether.


(a) Streamline Boustrophedon Path

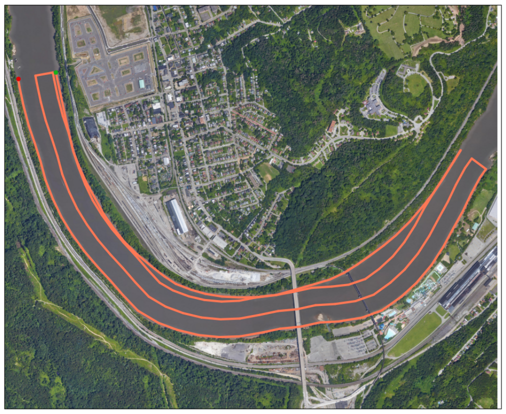

(b) Biased Streamline Boustrophedon Path

Figure 4.4: Comparison of coverage paths

In order to quantify the effects of our proposed optimization on the final coverage paths for this scenario, both the original and biased sets of path constraints were assembled into complete coverage plans using the streamline boustrophedon planner. The resulting streamline boustrophedon paths for each set of constraints are shown side by side in figure 4.4, and several relevant path metrics are presented in table 4.1. By applying the bias towards the riverbank with highest curvature, the length of the final coverage plan was reduced by about 3780 meters, which corresponds to a reduction of approximately 20.50%. Similarly, the total turns along the coverage path were reduced by about 258 degrees, or approximately 17.05%. These improvements were achieved while maintaining a 100% coverage rate across both plans.

Path	Length (m)	Coverage Rate	Total Turns (degrees)
Unbiased Streamlines	18440	100%	1515
Biased Streamlines	**14660**	100%	**1257**

Table 4.1: Comparison of Biased and Unbiased Streamline Coverage Paths

Although the reduction in length and total turns along the coverage path due to our overlap biasing optimization is clear, the benefits due to slower river currents near the inner bank of the river are not obvious since the back-and-forth nature of the streamline boustrophedon plan results in vehicle motion that alternates between movement with and against the currents. Depending on the direction the coverage path plan is executed in, our overlap biasing optimization may end up decreasing the energy cost of on leg of the journey while increasing the cost of another. To truly take advantage of the new distribution of path constraints in low-flow regions, the direction of travel along all constraints could be restricted so that the vehicle tends to move upstream in zones where the currents are slow and downstream at points where the currents are running quickly. This idea is crux of our last proposed optimization, which is the topic of the next section.

4.3 Flow-Aware Constraint Refinement and Sequencing

Although streamline constraints can be determined using the surface current field of a river, up until this point our proposed optimizations have been usable without any a priori knowledge of the river current distribution. Our final optimization bucks this trend as it requires at least some estimate of the ambient flow field throughout the target region in order to function. At its core, this optimization considers a set of undirected path constraints and a velocity flow field spanning the target region, and uses simulated travel times to impose a direction of travel along each constraint.

In order to achieve the desired result, the implementation of this optimization spans both the refinement and sequencing stages of the CB-CPP process. In the refinement component, we first determine a vehicle speed to use throughout our simulations by taking the maximum current velocity that occurs across the target region and increasing this value by 10%. The exact factor by which the simulated vehicle speed should exceed the maximum surface current magnitude is not critical, as long as the resulting speed used for simulation is guaranteed to exceed any flow speed that may be encountered within the target region. If this rule is not enforced, the simulation may get stuck as the vehicle fails to make progress against currents that exceed its prescribed velocity.

After selecting a suitable speed, the refinement module proceeds to simulate the motion of a vehicle at this speed along each constraint in both directions, noting the execution times. During simulation, the vehicle maintains a constant speed and is able to change directions instantaneously. The chosen vehicle speed can be thought of as the terminal velocity of the vehicle corresponding to a fixed thrust. If these simplifying assumptions are made, the simulated execution times can serve as reasonable surrogates for the energy usage by the vehicle required to drive the constraint.

Once the execution times have been determined, the refinement module defines an ″opportunity cost″ for each constraint for a given direction as follows: the cost for a constraint in the specified direction is equal to its execution time in that direction less its execution time in the opposite direction. Next the algorithm proceeds to consider the opportunity cost for all remaining undirected constraints in the forward direction, before selecting the constraint with lowest cost and assigning it the forward execution direction. Then the algorithm repeats this process, this time considering the opportunity costs of each undirected constraint in the reverse direction. If there are an even number of constraints in total, this process continues, alternating between forward and reverse directions, until all constraints have been assigned a direction. If there are an odd number of constraints, the process is the same, but for the last undirected constraint, the opportunity cost is computed in both directions and the constraint is assigned the direction corresponding to the lowest cost.

After refinement has been completed, each constraint should have an assigned direction and the number of ″forward″ constraint should differ from the number of ″reverse″ constraints by at most one. At this point the sequencing component of this optimization takes over to determine an efficient ordering of the constraints. Similarly to the greedy sequencers described in our CB-CPP case studies, the sequencer repeatedly selects the closest constraint using a specified heuristic to compute the cost from the previous constraint's egress point to the ingress points of all available constraints. In a departure from the typical greedy approach, however, this sequencer restricts the available constraints during the search to only the unsequenced constraints with a prescribed direction that opposes the direction of the previous constraint. This modification ensures that the sequencer doesn't select two

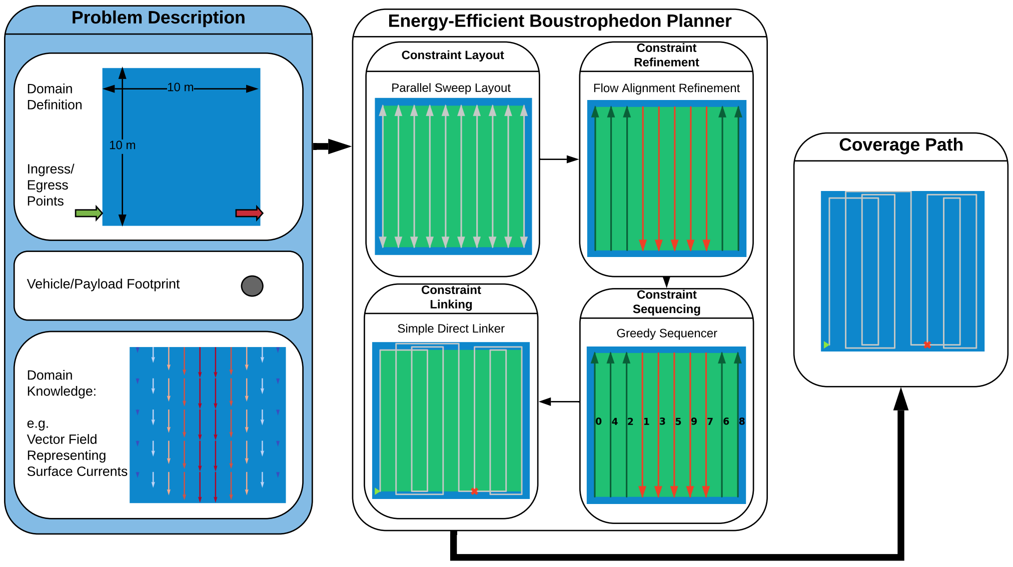


Figure 4.5: Flow-Aware Refinement and Sequencing

consecutive constraints that have the same execution direction, which could potentially force a vehicle to fight its way upstream against strong currents just to take advantage of those strong currents to traverse the next constraint. This situation is a particular problem when the width of the target region is significantly longer than its length, resulting in many shorter path constraints.

An example scenario that illustrates how this optimization can be combined with a conventional boustrophedon planner implemented within the CB-CPP to create an energy-efficient boustrophedon planner is shown in figure 4.5. The problem description block shown in blue has been extended to include the additional surface current domain knowledge required to employ the flow-aware optimization. Within the planner, we directly reuse the parallel sweep layout and simple direct linker components from our constraint-based boustrophedon implementation, and drop in the refinement and sequencing modules described in this section. In the example scenario, we define the surface current field using a model based on fully developed pipe flow, where the currents are strongest at the center of the flow and drop off inverse quadratically towards either bank. Examining the intermediate results and final output of the planner, we can recognize the flow-aware refinement imposing directions on the constraints that result in motion with higher speed currents at the center of the domain and against lower speed currents near the banks. The resulting coverage path resembles two adjacent spiral patterns on either side of the river that are oriented in opposite directions.

4.4 Summary

In this chapter we have presented three avenues for the optimization of coverage paths to improve energy-efficiency. First, we introduced the streamline constraint, a new type of path constraint with geometry defined by the streamlines of the river surface current field, which makes it particularly well-suited for covering diverse river geometries and enables the development of new optimizations due to their alignment with the river's flow. Next, we described the constraint coverage overlap biasing optimization, which takes advantage of river geometry to shift the layout of path constraints in a fashion that reduces their length and results in a denser distribution of path constraint in regions to the river with slower currents. Finally, we presented our flow-aware constraint refinement and sequencing optimization, which imposes a direction of travel along each constraint in a manner that minimizes the sum of simulated travel times along each constraint and results in as close to an even split of upstream and downstream constraints as possible. In the next chapter we introduce our simulation platform and explore several scenarios in simulation to evaluate our proposed optimizations.

Chapter 5

Simulation Experiments

In order to demonstrate the constraint-based coverage path planning architecture and validate the performance of the energy-efficiency optimizations described in the preceding chapter, we conducted various experiments both in simulation and in the field. Due to the difficulty in locating suitable scenarios in the real-world with sufficient variation in surface current data that was generally stable and easily measured, simulation experiments were critical in the development and validation of our approach. In this chapter we describe our simulation methodology and present simulation results from a variety of scenarios.

5.1 Simulation Platform

The simulation platform used in this work is a custom solution that simulates the motion of a surface vehicle through a region of fluid, which may itself be in motion defined by a vector field. Several simplifying assumptions are made in order to reduce the complexity of the simulation as well as its reliance on accurate models of the vehicle's intrinsic characteristics and dynamics; namely, that the vehicle moves at a constant speed and that it can change direction of travel instantaneously. By incorporating these assumptions, our simulation can approximate the vehicle moving at its terminal velocity when propelled by a constant thrust, while ignoring the time that would be required to accelerate to this speed and maintain it when changing direction. In this manner, our platform can simulate a vehicle's motion along a coverage path to obtain the total execution time for that path at a fixed thrust, which can serve as a surrogate measure for the vehicle's energy consumption.

We provide pseudocode for our simple simulator implementation in algorithm 1. In our implementation, the simulator is configured with a specified timestep and arrival threshold distance, which regulate the length of each simulation iteration and define how close the simulated vehicle must get to each waypoint respectively. Using a given vector field defining the flow, a desired constant boat speed relative to the local river current, and a coverage path to test, the simulator can then approximate the vehicle's motion along this path to produce two arrays of timestamps and associated vehicle positions. A key component of this simulator involves computing the optimal heading for the vehicle, which will move it along the desired coverage path given the flow at the current location and the constant speed to be maintained. This computation is abstracted in the ComputeMove function in our pseudocode for simplicity.

Algorithm 1: Simple Simulator

```
Input: dt, arrival_dist, flow_field, boat_speed, path
Output: times, positions
/* Initialize internal variables */
curr_time = 0.0
curr_pos = path[0]
target_idx = 1
curr_target = path[target_idx]
/* Initialize output arrays */
times = [curr_time]
positions = [curr_pos]
/* Run simulation until last waypoint has been reached */
while target_idx < len(path) do
    if curr_target - curr_pos < arrival_dist then
        target_idx += 1
        curr_pos = curr_target
        curr_target = path[target_idx]
    else
        curr_flow = flow_field[curr_pos]
        boat_move = ComputeMove(curr_pos, curr_target, curr_flow, boat_speed)
        curr_time += dt
        curr_pos += boat_move * dt
        times.append(curr_time)
        positions.append(curr_pos)
return (times, positions)
```

5.1.1 Evaluation Metrics

In order to evaluate and compare the performance of various coverage paths, several path metrics were identified, including path length, total turns along the path, coverage rate, and execution time. The first three of these metrics are independent of boat speed and can be calculated one time per each path without running the simulation. The execution duration metric is a direct output of the simulator and must be determined from simulation at specific vehicle speeds. We now briefly describe each metric, providing details on how it is computed and what its consequences are.

Path Length

This metric is simply the total length of all the segments making up the coverage path. Using the notation illustrated in figure 5.1, the path length is defined by the formula $\sum_{i=1}^{n} d_i$. Although this is a common metric to optimize in path planning, when the path traverses an area with moving fluids, the shortest path length does not necessarily imply the shortest execution time at a given vehicle velocity. Likewise when traversing uneven terrain of

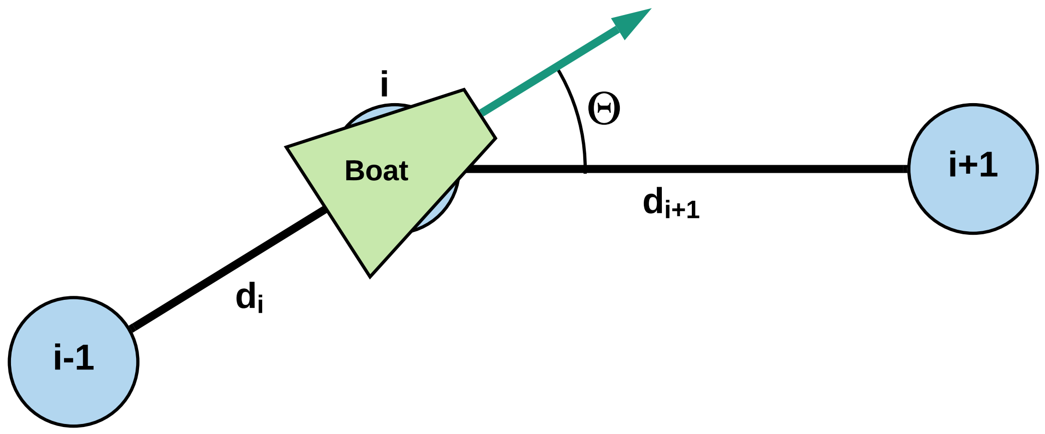


Figure 5.1: A diagram illustrating the computation of certain path metrics

varying elevation, the shortest path length will not guarantee the minimal energy cost of covering a given area. These are important considerations when planning for energy-efficiency and motivate the development of several other metrics to complement path length when evaluating coverage paths.

Total Turns

This metric is the summation of all the changes in heading the vehicle must perform while executing the target path. Using the notation from figure 5.1, the total turn metric is defined by the formula $\sum_{i=1}^{n-1} \theta_i$. This metric is useful to consider when evaluating the energy costs of a coverage path since increasing the amount of turns a vehicle makes will typically increase the total energy costs of executing the path. This stems from the fact that when a vehicle changes heading, it must accelerate along the vector corresponding to the difference of its previous velocity vector and its new velocity vector. In practical terms, this means the vehicle must arrest its previous motion in the direction perpendicular to its new heading and accelerate along its new heading to reach its desired speed. Although there are maneuvers specific to the vehicle that can mitigate the effects of turning, a change in heading will always incur some energy costs and is therefore important to keep in mind.

Coverage Rate

The coverage rate is defined as the total area of the polygon formed by sweeping the sensor/actuator envelope along the coverage path, divided by the total area of the region being covered. These components are illustrated in figure 3.9 in our discussion of layout modules in chapter 3. For complete coverage path planning, the goal is to keep this coverage rate as close to 100% as possible, and although additional energy savings are undoubtedly achievable by sacrificing coverage rate, that is beyond the scope of our objectives/application. Therefore, for this work we will simply use this metric as a check to verify that our coverage path achieves complete coverage, rather than a means of evaluation. This metric is nevertheless important across all coverage planning work.

Execution Time

As we alluded to when describing the path length metric, length alone does not always correspond to the true time or energy costs for a vehicle executing the path, depending on the domain environment. In order to approximate the true time a path will take to execute, we rely on our simulator, which takes into account fluid motion throughout the domain. This metric is important because it provides an execution time for a path given a constant velocity, which can represent the terminal velocity at a constant thrust. It therefore makes a reasonable surrogate for energy cost since at any fixed thrust, a path which takes longer to execute will generally have a higher energy cost.

5.2 Results

In this section we present the results from several simulation experiments using the platform described in the previous section. For each experiment, we first describe the scenario and the reasoning behind our design decisions, then show the test and baseline coverage paths generated by our planner and a conventional boustrophedon approach, and finally present simulation data to evaluate the performance of our energy-efficient plans. Because a typical boustrophedon path can be assembled in two different ways from any given set of transects (i.e., each transect can be connected to the following adjacent transect at either of its endpoints), we consider both possibilities when determining the baseline performance. Furthermore, we additionally simulate execution of each baseline path in both directions, and use the best performing combination to compare against the path generated by our proposed planner. This ensures any observed benefits of our method are independent of how a competing boustrophedon planner may be configured.

For the sake of a clear and compact presentation of our results, only one of the possible baseline boustrophedon paths is illustrated alongside the energy-efficient paths throughout this discussion. In addition, only the best performing boustrophedon baseline data is presented in figures and tables within this section. For more detailed tables that include complete baseline simulation results, refer to appendix B.

5.2.1 Four Transect Field Experiment Analog

Our first scenario was designed to simulate a section of river adjacent to its left bank extending partway into the channel. The target coverage area and surface current model for this scenario were chosen to approximate our four transect field experiment on the Cape Fear River that we present in chapter 6, in order to provide an interesting comparison between simulation and real-world experiments. These features are reflected in figure 5.2, where the blue polygon denotes the target region and the colored vector field represents the surface currents.

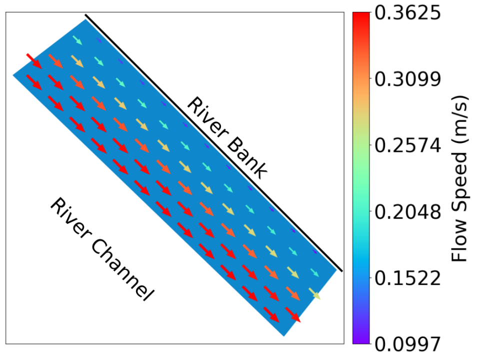
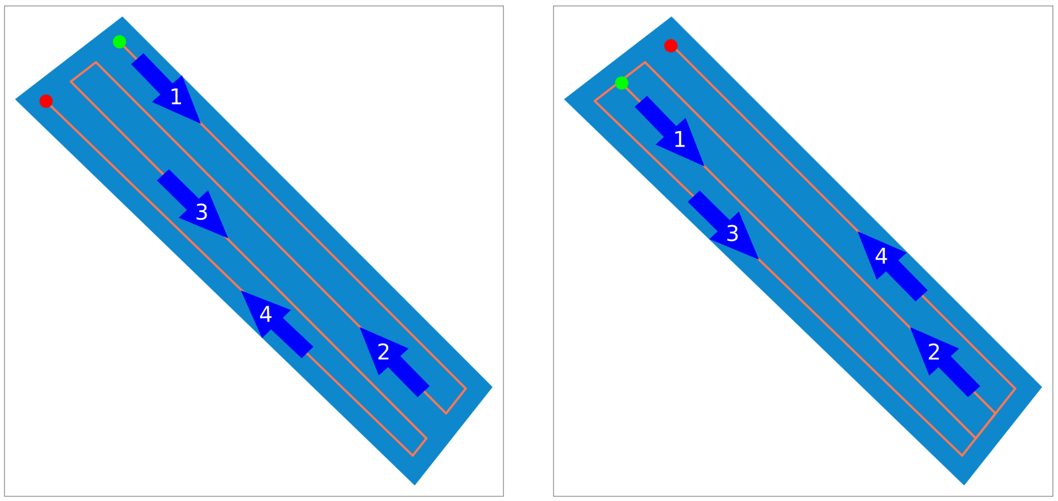


Figure 5.2: Field experiment analog scenario

The target region for this scenario has a total area of about 3580 m^2 with a current distribution that is fastest towards the center of the river channel, reaching a maximum speed of approximately 0.36 m/s and then dropping off quadratically towards the river bank. The minimum surface current speed within the region is 0.082 m/s, while the mean and median flow speeds are 0.29 and 0.32 m/s respectively. This skew of the mean and median towards the upper end of the range of current speeds within the region implies that a greater proportion of the area has relatively high currents, a fact borne out by the vector field plot in figure 5.2. As a result, achieving energy-efficiency will rely in large part on exploiting the region closest to the bank where the current is slowest in order to move upstream during coverage.



(a) Boustrophedon Coverage Path (b) Energy-Efficient Coverage Path

Figure 5.3: Baseline and test paths for the field experiment analog scenario

In this scenario, we assess the performance of our proposed coverage planner with the flow-aware refinement and sequencing optimization applied as described in the previous chapter. We evaluate the energy-efficient coverage path produced by our planner against a baseline coverage path produced by a conventional boustrophedon planner. The baseline and test coverage paths for this experiment are shown side by side in figure 5.3, with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.

Path	Length (m)	Coverage Rate	Total Turns (degrees)
Boustrophedon	**472**	99.17%	**539**
Energy-Efficient	494	**99.39%**	540

Table 5.1: Field Experiment Analog Scenario Coverage Path Metrics

Upon examination of several of our coverage path evaluation metrics, presented in table 5.1, we can see that the length of the energy-efficient coverage path produced by our planner is about 22 meters, or about 4.66%, longer than the boustrophedon baseline. This is an expected consequence of sequencing the execution of the transects that comprise both coverage paths because our proposed planner at times chooses to travel transects that may be further away but have more advantageous surface currents for the next leg of the path. A small benefit of this extra distance is that the energy-efficient paths produced by our planner tend to achieve better coverage rates due to this additional length, which is also the case in this scenario. Finally, the total turns in each path are nearly the same, with the 1 degree difference being attributable to the irregular nature of the target area polygon.

In order to gauge the energy requirements for each coverage path, we used our simulator to estimate the execution times for a vehicle moving at a range of fixed speeds. Due to the design of our simulation platform, we must restrict the vehicle speed to lie above the maximum surface current speed within the target region in order to avoid situations where the simulated vehicle gets stuck heading directly into the currents or cannot successfully move along the planned path. Therefore, the range of simulated vehicle speeds was fixed at 0.375 to 1.0 m/s, in increments of 0.025. A plot of the simulated path durations vs. the simulated boat speed is presented in figure 5.4.

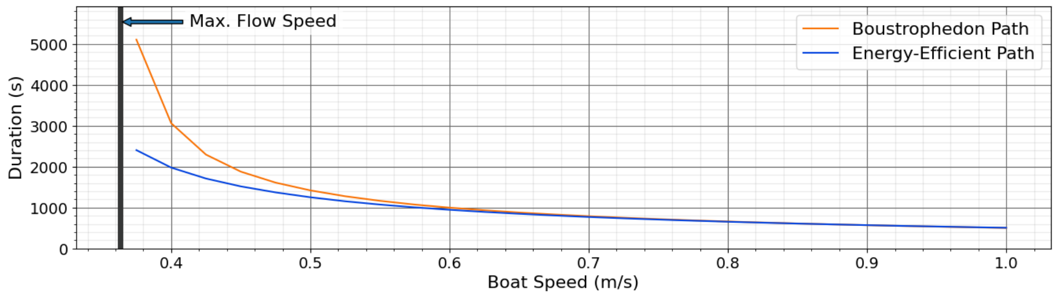


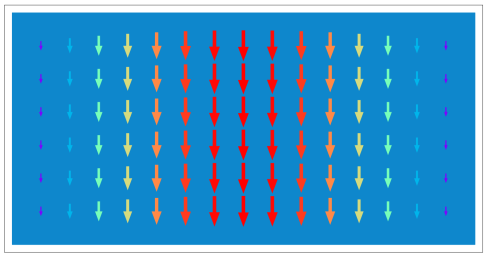
Figure 5.4: Field experiment analog scenario simulation results

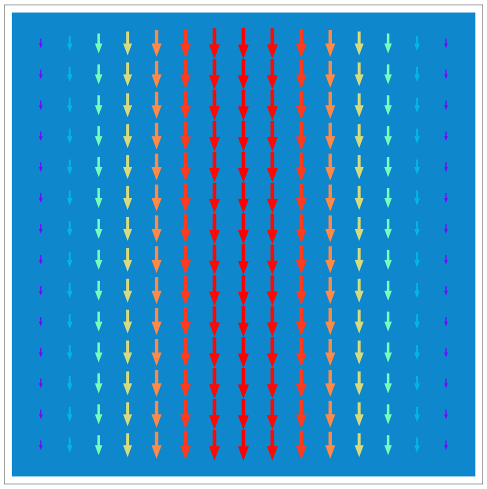
From this visual representation of our results, it is evident that the energy-efficient path takes substantially less time to execute when the vehicle is travelling at a fixed speed that is close to the maximum surface current speed within the target region; at an operating speed of 0.375 m/s, the energy-efficient approach results in a 54.92% decrease in execution time over the best possible boustrophedon path. However, this effect diminishes as the speed of the vehicle increases relative to the surface currents within the target region. At an operating speed of 0.9 m/s and above, the duration curves in figure 5.4 cross, indicating that the energy-efficient path actually performs very slightly worse than the best boustrophedon path.

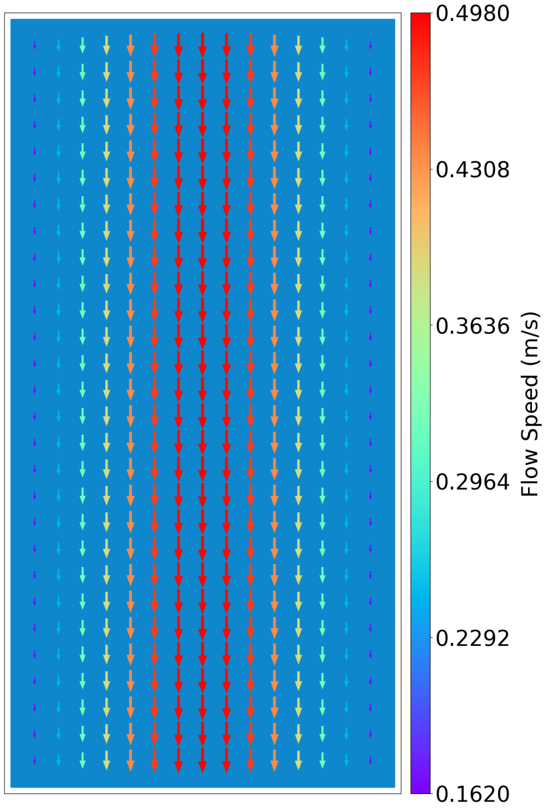
Since the energy-efficient path is over 4% longer than the boustrophedon path, this explains how the boustrophedon pattern is able to eventually beat out our proposed energy-efficient approach at higher vehicle speeds. In fact, we would expect the difference in execution times between the two paths to trend towards that same 4% differential as the vehicle speed continues to increase and the surface currents become increasingly less relevant. A complete table of simulation results for this experiment can be found in appendix B.

5.2.2 River Cross-Section

In this scenario we simulate several cross-sections of a river with a unidirectional surface current distribution. The magnitude of this current field is modelled after the velocity profile of fully developed pipe flow [31], with a maximum flow speed of 0.5 m/s occurring at the center of the channel and then dropping off inverse quadratically towards both banks of the river. The width of the river is fixed at 150 meters, while the lengths of the cross-sections considered are 75, 150, and 300 meters, resulting in target regions with areas of 11250, 22500, and 45000 m^2 respectively. The geometry and surface current distribution of each cross-section is shown in figure 5.5.


(a) Wide River Cross-Section


(b) Square River Cross-Section

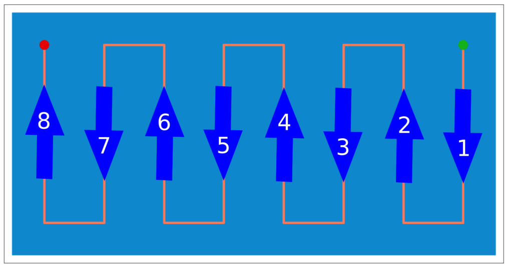


(c) Long River Cross-Section

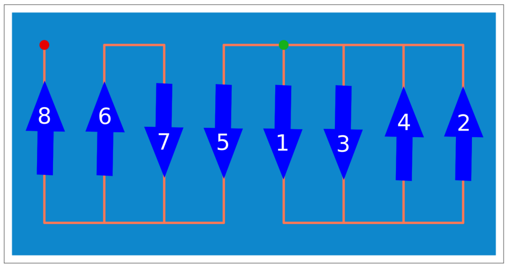
Figure 5.5: River cross-section scenario

Like the previous experiment, this scenario was designed to evaluate the performance of our proposed coverage planner with the flow-aware refinement and sequencing optimization applied. The reasoning behind considering three different lengths of river cross-section together in this experiment, however, was to assess the impact of the transect length on the energy-efficiency achieved by our planner. To isolate this experimental parameter, the transect orientation was chosen to lie parallel to the river's center line and flow, the river width was fixed, and the surface current distribution was held constant. This resulted in a set of 8 transects for each cross-section geometry, which were colinear and differed only in length.

The computed transects for each cross-section were assembled into coverage paths by our planner and a baseline boustrophedon coverage planner, after which the execution of these paths was simulated with our simulation platform. The baseline and test coverage paths for the wide river cross-section are shown side by side in figure 5.6, with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.



(a) Boustrophedon Coverage Path



(b) Energy-Efficient Coverage Path

Figure 5.6: Baseline and test paths for the wide river cross-section scenario

Because the coverage paths determined for the square and long river cross-sections are simply elongated versions of the paths shown in figure 5.6 that share the same transect sequence and endpoints, we have omitted dedicated figures for these paths in this section. These figures are made available in appendix B.

An examination of our coverage path evaluation metrics, presented in table 5.2, reveals that the energy-efficient coverage path produced by our planner is consistently 130 meters longer than the baseline boustrophedon path, regardless of the cross-section region. This is an expected result since the additional length of the energy-efficient paths is solely a consequence of additional travel between transects, and the route between transects does not change for either path across the scenario variants.

We also note that the coverage rate of each path tends to increase as longer cross-sections of river are considered. This increase can be attributed to the fact that any gaps in coverage occur at the domain boundaries where transects are joined together and do not change as total scenario area grows. Finally, the total turns along all paths across regions are equal, which is unsurprising considering that the transect orientation is fixed and all the transects for each scenario variant are the same length; practically this implies that the vehicle will need to execute two 90 degree turns between execution of any two transects.

Region	Path	Length (m)	Coverage Rate	Total Turns (degrees)
Wide	Boustrophedon	**570.00**	97.22%	1260.00
Wide	Energy-Efficient	700.00	**98.37**%	1260.00
Square	Boustrophedon	**1170.00**	98.61%	1260.00
Square	Energy-Efficient	1300.00	**99.18**%	1260.00
Long	Boustrophedon	**2370.00**	99.30%	1260.00
Long	Energy-Efficient	2500.00	**99.59**%	1260.00

Table 5.2: River Cross-Section Scenario Coverage Path Metrics

As in our previous experiment, we used our simulator to estimate the execution times for a vehicle moving along the test and baseline paths at a range of fixed speeds. Since the maximum flow speed that occurs in this scenario is 0.5 m/s, the range of simulated vehicle speeds was restricted between 0.51 and 1.0 m/s to avoid badly conditioned situations within the simulator. The simulated path durations for each path type and cross-section geometry are plotted against the simulated boat speed in figure 5.7, where the boustrophedon baseline path results and energy-efficient path results are denoted in orange and blue respectively. Additionally, the continuity of the lines is used to indicate the corresponding cross-section for each path in the plot legend.

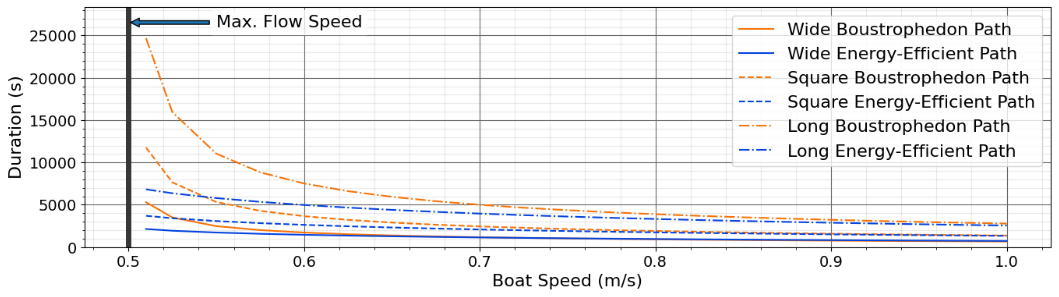


Figure 5.7: River cross-section scenario simulation results

This visual representation of our results reveals a significant reduction in execution time for the energy-efficient paths when the vehicle is travelling at a fixed speed that is close to the maximum surface current speed within the target region. This effect diminishes as the speed of the vehicle increases. Comparing the data for paths covering the different cross-sections, it is clearly evident that the energy saving benefit from our proposed flow-aware refinement and sequencing optimization is substantially magnified as the length of the transects making up the coverage path increases. This conclusion is borne out in the complete simulation data, which is available in appendix B.

5.2.3 River Bend Scenario

In this scenario we model a section of river as it rounds a 180 degree bend. Bends in a river make for an interesting scenario to test our planning algorithm, as the change in direction gives rise to inertial forces that drive the high-velocity core of the river towards the outer bank, thereby creating an asymmetric current distribution throughout the region [55]. Furthermore, this asymmetric flow results in erosion at the outer bank and deposition at the inner bank, which can cause significant changes in the river channel width along the curve [21]. These characteristics are captured in our scenario design as shown in figure 5.8.

The target region in this scenario represents the entire channel of a river as it flows around the bend, encompassing a total area of approximately 62780 m^2. The channel width varies from a minimum of 100 meters at the narrowest points upstream and downstream from the bend, to a maximum width of 150 meters at the midpoint of the bend. For the surface current distribution, we use the same fully-developed pipe flow model as for the previous scenario, but modify it to shift the center axis of flow along the river bend. While the center axis starts out centered within the river channel upstream of the bend, the axis is biased

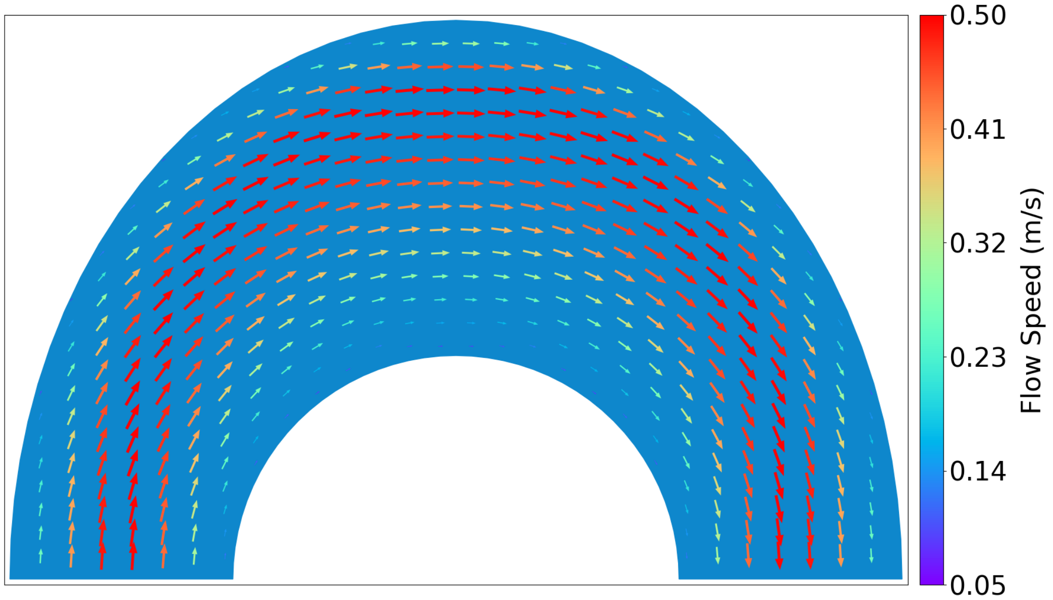


Figure 5.8: River bend scenario

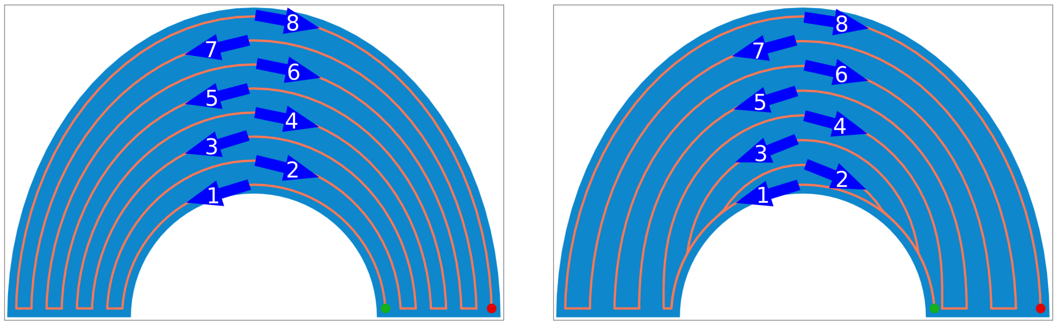
towards the outer bank throughout the curve to simulate the inertial effects that occur at real river bends. As before, the maximum flow speed is 0.5 m/s at the center axis and drops off inverse quadratically towards each bank.

In our experiments for this scenario, we evaluate the performance of our proposed coverage planner with the coverage overlap bias and flow-aware energy-efficiency optimizations, which we described in the previous chapter. Contrary to the previous scenarios discussed up to this point, the baseline planning method for these experiments is the streamline boustrophedon technique introduced in a case study in chapter 3. This approach is better suited to coverage of winding river domains, where a conventional boustrophedon planner would require a preliminary cellular decomposition step and may not achieve as good a coverage rate due to the irregular transects that may be generated. Furthermore, because the transects of a boustrophedon path will likely not be aligned with direction of flow in the river across the target region, it limits the effectiveness and extents to which our energy-efficient sequencing optimization can be applied.

The remainder of this section lays out the results of our simulation experiments. We begin by examining the effect of our coverage overlap bias optimization when applied to the streamline boustrophedon coverage pattern and evaluate how the resulting plan performs against the baseline. In the next experiment, we repeat the same analysis with a path that was planned with our flow-aware optimization applied. In the third experiment, we demonstrate the layerable nature of CB-CPP algorithms by considering the performance of our planner when both the coverage overlap bias and flow-aware optimizations are applied. Finally, we conclude our discussion of this scenario by comparing the best-performing energy-efficient path produced by our planners against a conventional boustrophedon coverage path in order to address any concerns that the proposed approach only performs well relative to the streamline boustrophedon coverage pattern.

Constraint Coverage Overlap Bias

In our first experiment, we investigate the benefits of our proposed constraint coverage overlap bias optimization. Since the scenario domain narrows both upstream and downstream of the river bend, coverage transects oriented along the streamlines of the river will necessarily have some overlap in these regions. In the baseline case, the transects are evenly spaced along each cross-section of the river; however, by biasing transects towards the inner bank of the river bend, the total length of all coverage transects, and consequently the complete coverage path, can be reduced. The baseline and test coverage paths for this experiment are shown side by side in figure 5.9, with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.



(a) Streamline Coverage Path (b) Biased Streamline Coverage Path

Figure 5.9: Exploiting coverage overlap to enhance coverage path plans

The coverage path metrics for both the baseline and test paths are presented in table 5.3. Looking at the path lengths, we note that the coverage overlap biasing does indeed shorten the path as expected, by about 170 meters, or 4.19%. At the same time, the coverage rate only marginally decreases by 0.08%, while the total turns along the path increase by approximately 142 degrees, or 5.31%. This turn increase is a consequence of the biasing deforming several transects and could be partially remedied by applying some smoothing during the planning process.

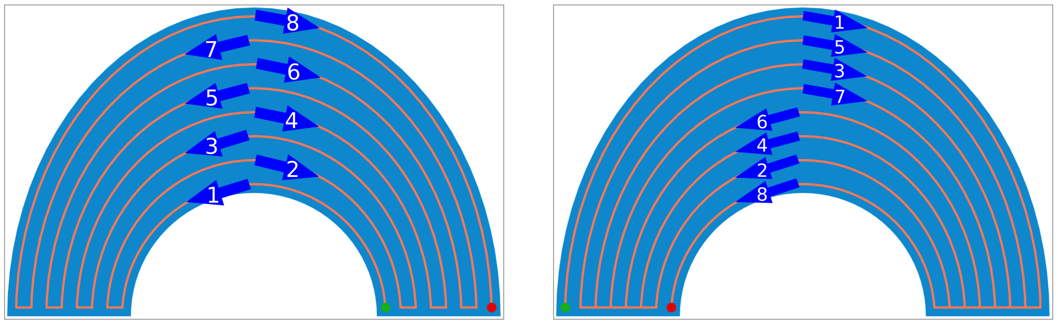
Path	Length (m)	Coverage Rate	Total Turns (degrees)
Baseline	4060	**99.99**%	**2672**
Biased Streamline	**3890**	99.92%	2814

Table 5.3: Baseline vs. Biased Streamline Coverage Path Metrics

When vehicle execution of these paths is simulated at various fixed speeds and the total travel time is recorded, the advantage of the biased coverage plan is even more apparent; the performance gain ranges from a 32.71% reduction in execution time when the vehicle speed is fixed at 0.51 m/s, to a 6.95% reduction at 1.0 m/s travel speed. The simulated execution times for the full range of vehicle speeds is provided in appendix B.

Flow-Aware Constraint Refinement and Sequencing

In our next experiment, we consider the flow-aware optimization and evaluate its impact when applied to our proposed streamline boustrophedon planner. Because the surface currents in this scenario are faster towards the outer bank of the river bend, we would expect this optimization to prioritize upstream travel along the transects closest to the inner bank. A closer examination of the coverage paths for this experiment presented in figure 5.10 confirms this hypothesis. In this figure, we present the baseline and test coverage paths with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.



(a) Streamline Coverage Path (b) Energy-Efficient Coverage Path

Figure 5.10: Optimizing constraint direction and sequencing using flow information

The coverage path metrics for both the baseline and test paths are presented in table 5.4. Looking at these metrics, we note that the flow-aware optimization extends the length of the energy-efficient coverage path by approximately 230 meters, or 5.67%, while maintaining the same 99.99% coverage rate as the baseline path. The total turns along each path are about the same, which is expected since both paths are built from the same set of constraints.

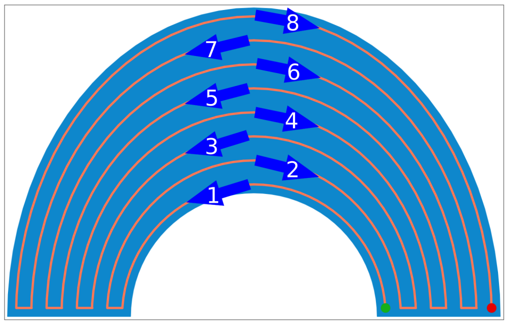
Path	Length (m)	Coverage Rate	Total Turns (degrees)
Baseline	**4060**	99.99%	2672
Energy-Efficient	4290	99.99%	2672

Table 5.4: Baseline vs. Energy-Efficient Coverage Path Metrics

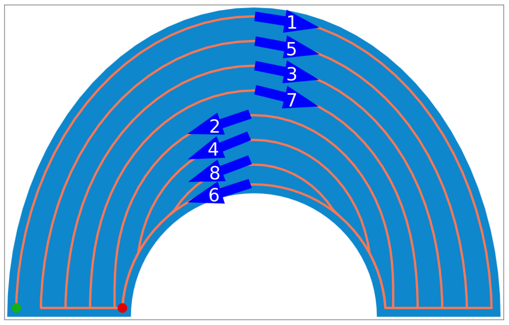
As in our previous experiment, we used our simulator to estimate the execution times for a vehicle moving along the test and baseline paths at a range of fixed velocities between 0.51 m/s and 1.0 m/s. The results from these simulations reveal a significant reduction in path execution time when the vehicle is travelling slower speeds close to the maximum surface current speed within the target region. At a fixed vehicle speed of 0.51 m/s, this improvement amounts to 54.11%, but this is substantially reduced to 5.83% when the vehicle speed is increased to 1.0 m/s. The simulation results for the full range of vehicle speeds for this experiment are provided in appendix B.

Combined Optimizations

In our third experiment, we demonstrate the layerable nature of the CB-CPP framework by applying both the coverage overlap bias and flow-aware energy-efficiency optimizations sequentially within our proposed planner. By combining these two methods, we hypothesize that the observed improvement in resulting path execution time will be at least match the sum of the improvements we observed when each optimization was applied individually. The baseline and test coverage paths for this experiment are shown side by side in figure 5.11, with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.



(a) Streamline Coverage Path



(b) Biased Energy-Efficient Coverage Path

Figure 5.11: Multiple optimizations to enhance coverage path plans

Upon examination of the coverage path metrics presented in table 5.5, we find that the energy-efficient path produced by our proposed planner with both optimizations applied scores worse than the baseline path across all the metrics. The length of our path is about 90 meters, or 2.22% longer than the baseline path, which lies in between the 4.19% decrease we observed when applying the coverage overlap bias alone, and the 5.67% increase observed due to the energy-efficient sequencing. The coverage rate is virtually unchanged between the test and baseline paths, while the total turns along the energy-efficient path are increased by approximately 152 degrees.

Path	Length (m)	Coverage Rate	Total Turns (degrees)
Baseline	**4060**	**99.99%**	**2672**
Biased Energy-Efficient	4150	99.98%	2824

Table 5.5: Baseline vs. Biased Energy-Efficient Sequencing Coverage Path Metrics

Despite our test path performing worse on every metric we examined, the simulation of path execution at different vehicle speeds reveals an entirely different result; at a fixed speed of 0.51 m/s, the energy-efficient path took 76.49% less time to execute than the baseline path. Even at the higher vehicle speed of 1.0 m/s, our test path still produced a 15.60% improvement in execution time over the baseline. The simulated execution times for this energy-efficient path are plotted in figure 5.12 and a complete table of numerical results is provided in appendix B.

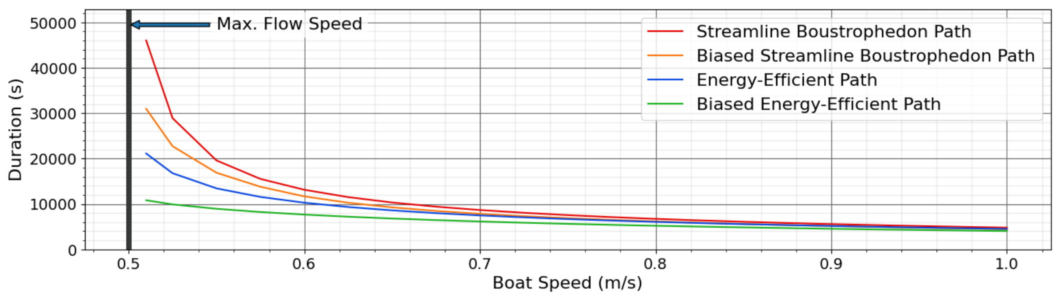


Figure 5.12: Coverage path execution times for various vehicle velocities

In addition to the results from this experiment, figure 5.12 presents the simulated durations for paths planned with each optimization individually alongside the duration of the baseline streamline boustrophedon path. This visual representation shows that applying any of our proposed optimizations results in a performance improvement over the baseline method, and applying both optimizations together yields even greater improvements. Table 5.6 lists the percentage improvement resulting from each proposed optimization individually, the sum of these improvements, and the percentage improvement resulting from the combined optimizations.

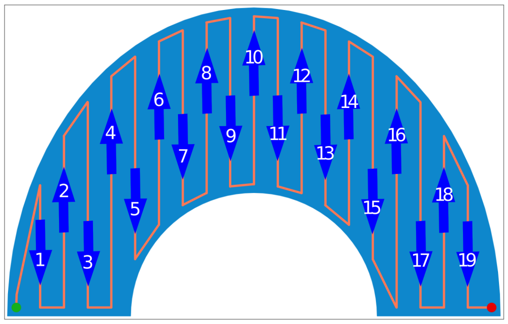
Vehicle Speed (m/s)	Coverage Overlap Bias	Flow-Aware	Sum of Improvements	Combined Optimizations
0.510	32.71%	54.11%	86.82%	76.49%
0.525	21.41%	41.97%	63.37%	65.75%
0.550	13.81%	31.42%	45.23%	54.39%
0.575	11.16%	25.58%	36.74%	47.00%
0.600	11.09%	21.72%	32.81%	41.64%
0.625	10.76%	18.93%	29.70%	37.50%
0.650	10.37%	16.80%	27.17%	34.19%
0.675	9.97%	15.10%	25.07%	31.45%
0.700	9.59%	13.70%	23.29%	29.15%
0.725	9.24%	12.52%	21.77%	27.18%
0.750	8.92%	11.51%	20.44%	25.46%
0.775	8.63%	10.64%	19.27%	23.96%
0.800	8.37%	9.86%	18.23%	22.62%
0.825	8.13%	9.18%	17.31%	21.43%
0.850	7.91%	8.56%	16.47%	20.35%
0.875	7.72%	7.99%	15.71%	19.38%
0.900	7.53%	7.49%	15.02%	18.49%
0.925	7.37%	7.02%	14.39%	17.68%
0.950	7.22%	6.59%	13.81%	16.93%
0.975	7.08%	6.20%	13.27%	16.24%
1.000	6.95%	5.83%	12.77%	15.60%

Table 5.6: Comparison of Execution Time Improvements

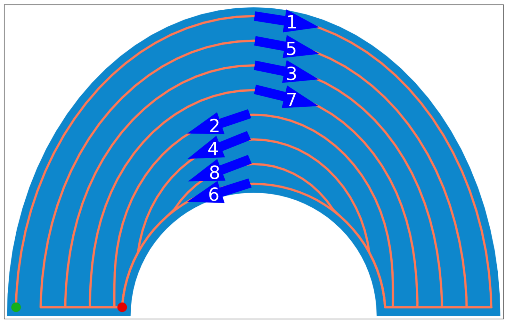
If we compare the percentage improvement in path execution time produced by our energy-efficient path for this experiment and compare it against the improvements seen in the previous experiments, an interesting pattern emerges. Not only does the combined planner generally produce an improvement that is comparable to the sum of the improvements observed when each optimization is applied individually, but the improvement actually exceeds this sum for most simulated vehicle velocities. This suggests that there is some synergistic effect taking place between the two optimizations that results in further reduced execution time than what can be expected from each optimization alone. For example, it may be that biasing transects towards the inner bank of the river bend creates shorter routes that the energy-efficient sequencing optimization then exploits to move upstream more quickly than was possible without the biasing.

Conventional Boustrophedon Comparison

In the previous section we presented the results of our proposed planner with coverage overlap biasing and flow-aware energy optimization on the river bend scenario and showed that it resulted in a significant reduction in execution time when compared the the baseline planning method - the streamline boustrophedon planner. Although this planner is well suited to river domains, the paths it produces tend to be longer than a conventional boustrophedon approach may yield. Therefore, in this section we repeat our previous analysis, comparing the biased energy-efficient path against a new baseline, the conventional boustrophedon path. The baseline and test coverage paths for this experiment are shown side by side in figure 5.13, with the sequence of transects annotated with blue arrows for clarity. The start and end points of each path are denoted with green and red circles respectively.



(a) Conventional Boustrophedon Coverage Path



(b) Biased Energy-Efficient Coverage Path

Figure 5.13: Comparison against a conventional boustrophedon plan

The path metrics for both the conventional boustrophedon path and our proposed energy-efficient path are presented in table 5.7. Due to a greater extent of coverage overlap along its route, the energy-efficient path is approximately 780 meters longer than the convention boustrophedon path, which corresponds to a substantial 23.15% increase over the baseline. At the same time, the coverage rate of the conventional boustrophedon pattern is inferior to the energy-efficient path because of the discretization that occurs along the smooth banks of the river during the layout of straight line transects throughout the region. This results in uncovered parts of the target region, as seen in the bottom right of figure 5.13a, between transects 15 and 16 as well as to the right of transect 19.

Path	Length (m)	Coverage Rate	Total Turns (degrees)
Conventional Boustrophedon	**3370**	97.18%	3600
Biased Energy-Efficient	4150	**99.98%**	**2824**

Table 5.7: Boustrophedon vs. Biased Energy-Efficient Path Metrics

By replacing the many straight line transects of the conventional boustrophedon plan with a smaller number of longer curved path segments that approximately follow the streamlines of the river, the energy-efficient path produced by our proposed planner can achieve better coverage of the target region while at the same time significantly reducing the total turns along the coverage path. Compared to the conventional boustrophedon path, our optimized path turns 776 degrees less along its route, which amounts to a decrease of 21.55% from the baseline performance.

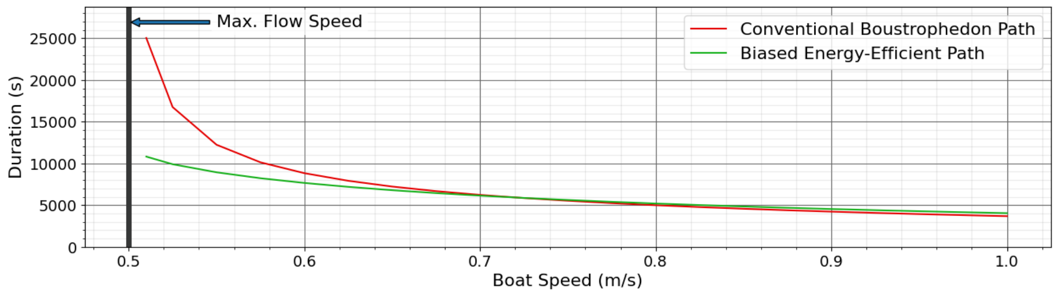


Figure 5.14: Comparison against conventional boustrophedon coverage path

As in our previous experiments, we used our simulator to estimate the execution times for a vehicle moving along the test and baseline paths at a range of fixed velocities between 0.51 m/s and 1.0 m/s. The simulated path durations for each path type are plotted against the simulated boat speed in figure 5.14, where the conventional boustrophedon baseline path results and energy-efficient path results are denoted in red and green respectively. Despite the 23.15% longer length of the energy-efficient path, these simulation results once again reveal a significant reduction in execution time for the energy-efficient path over the conventional boustrophedon baseline path when the vehicle is travelling at fixed velocities close to the maximum surface current velocity within the target region. At a simulated boat speed of 0.51 m/s, our energy-efficient approach reduces the total time required to cover the target area by nearly 4 hours, or approximately 56.80% This effect diminishes as the speed of the vehicle increases, and at speeds upwards of 0.725 m/s the baseline begins to outperform the energy-efficient path. The full set of simulated execution times for this experiment is provided in appendix B.

5.3 Summary

In this chapter we described the simulation platform that was created specifically to evaluate the performance of the energy-efficient planners developed in this work, laid out several metrics that can be used to compare different aspects of coverage paths, and presented simulation results for several scenarios to test our proposed planning methods. Through our analysis we showed that the energy-efficient optimization presented in this work can significantly reduce the time required for a vehicle moving at a constant speed to cover a given region in the presence of flow. Since travel at a constant speed can essentially be thought of as movement at the terminal velocity achieved by a fixed thrust, and the power consumption for motors outputting a fixed thrust will also remain relatively constant, it is reasonable to conclude that the execution time for a path will be directly related to the energy used by the vehicle while executing this path at a fixed speed. This coupled with the very significant performance improvements we observed across our scenarios, strongly suggest that the optimization methods proposed in this work will significantly reduce energy usage during coverage, particularly when the vehicle is moving at speeds close to the surface current velocities within the target region. In the next chapter, we will present results from the field that validate this hypothesis.

Chapter 6

Field Experiments

As we alluded in the description of our simplified simulation methodology in the previous chapter, accurately modelling fluid dynamics in order to perfectly simulate an autonomous surface vehicle's motion and energy consumption during operation is a challenging endeavor that is seldom realistic. Although we developed and described several indicators/metrics and simplifications for our simulation in an attempt to overcome these difficulties, actually deploying an ASV in the field and directly measuring energy costs of various coverage patterns is indispensable to validate our constraint-based coverage path planning approach and evaluate the performance of the proposed energy-efficiency optimizations. In this chapter we introduce the test platform built for this work, present results from field experiments using this vehicle in North Carolina, and compare these results to simulation.

6.1 Test Platform

The test platform used for field experiments in this work is a purpose-built autonomous catamaran, measuring 1.5 meters in length and weighing approximately 25kg when outfitted with two batteries. The vehicle is built from two roto-molded HDPE plastic hulls, each containing a waterproof compartment that houses the computing, power distribution, and other electronics. The hulls are joined together with lengths of PVC tubing that are clamped to plastic plates fixed to each hull's deck. The clamping points along the tubing can be adjusted to conveniently change the vehicle width between deployments, but for the purposes of these experiments the boat's width was fixed at 1.1 meters at its widest point. An annotated image of the vehicle is shown in figure 6.1.

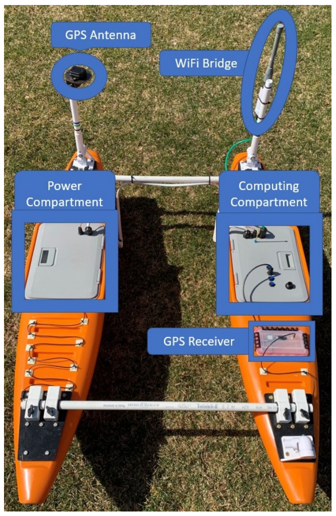
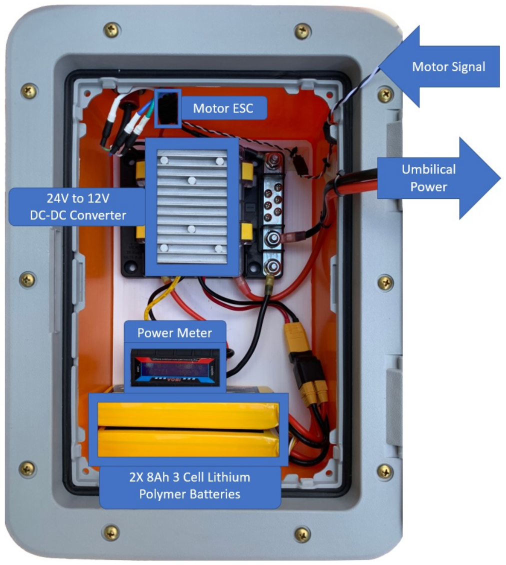
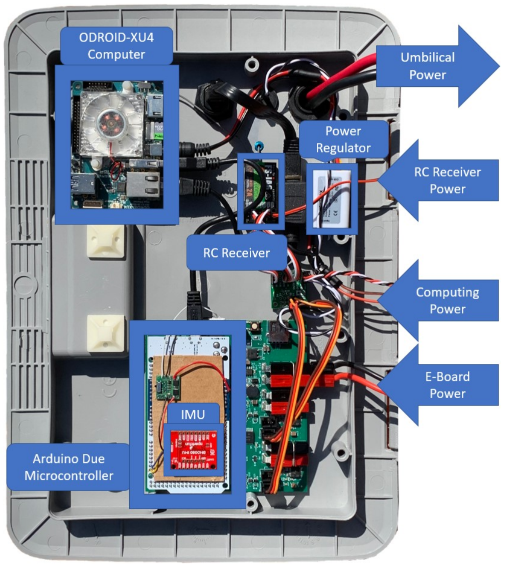


Figure 6.1: Field test platform

Each hull of the vehicle contains a waterproof compartment, two deck plates that serve as attachment points for the cross-beams, a collapsible antenna mast at the stern, and a T200 Blue Robotics thruster below the waterline. With the exception of a GPS receiver mounted to one hull's deck as


Motor Signal
Motor ESC
Umbilical Power
24V to 12V DC-DC Converter
Power Meter
2X 8Ah 3 Cell Lithium Polymer Batteries

(a) Power Compartment



(b) Computing Compartment

Figure 6.2: Hull Compartment Internals

well as the GPS and WiFi antennas mounted to the rear masts, the bulk of vehicle electronics are partitioned between the two hull compartments, hereafter referred to as the power and computing compartments. In order to reduce the effects of electromagnetic interference on sensitive electronics such as the inertial sensors used for navigation, the batteries and high current power regulator are isolated in the power compartment while the computing and sensors are housed in the computing compartment. Diagrams illustrating the most important components within each compartment are provided in figure 6.2.

The power for the system is provided by two 8Ah three-cell lithium polymer batteries connected in series to provide approximately 24 Volts. This voltage varies as the batteries discharge and additionally depends on the current being drawn from the batteries at any given time. Since the voltage supplied to the thrusters dictates the speed of at which they rotate for a specified command, these variations in battery voltage present a challenge for driving the thrusters at a fixed rate and evaluating the energy consumption of the vehicle. Without compensating for this effect, which also varies from battery to battery, the thrusters will run ever slower for a fixed motor command thereby reducing the speed of the vehicle as it executes its mission and the batteries are depleted. Because the Blue Robotics T200 thrusters on the test vehicle are more efficient at lower speeds and the drag forces on the displacement hulls grow as their velocity through a fluid increases, this reduction in speed generally results in more efficient operation as well. This makes a comparison of the energy required to execute two different path plans difficult, particularly when one path takes longer to drive than the other or different batteries are used across experiments.

In order to overcome this challenge, we make use of a 24V to 12V DC-DC buck converter connected in between the battery packs and the rest of the vehicle's power system. Although adding this component introduces some additional energy losses to the system, the output from the converter is a stable 12 Volts, which ensures the thrusters maintain a consistent speed at any given command throughout operation. Additionally, the system is now agnostic to the internal characteristics of the installed battery packs, which allows for a cleaner comparison between experiments conducted with different batteries and in turn enables rapid redeployment of the test platform with fresh batteries when collecting data in the field. This is a critical capability for our purposes because the experimental coverage paths we are testing rely on some knowledge of the surface current distribution, which can significantly over time in the field.

After passing through the regulator, power is routed through a wattmeter that measures the total energy consumed by all vehicle systems during operation, except for the losses introduced during voltage regulation. The reasoning behind this placement is that it avoids incorporating any irregularities that might occur during regulation into the energy measurement while improving the power meter performance by removing the need to account for voltage variation. After passing through the meter, power is distributed via a fuse block to the motor electronic speed controller (ESC) and the power umbilical, the latter of which carries power to the computing compartment within the second hull.

The computing compartment contains the bulk of the electronics and sensors responsible for the control and operation of the autonomous surface vehicle. Unlike the contents of the power compartment, the computing components are predominantly mounted to the hatch of the compartment as shown in figure 6.2b. This setup allows for convenient connections outside the compartment without passing wires through the molded hull and mitigates the risk of water damage if the compartment were to flood. The primary onboard computer is an ODROID XU4 device, which is powered by an ARM processor and runs Ubuntu 16.04. The ODROID interfaces with navigational sensors and motor controllers through its connection to an Arduino Due microcontroller with custom electronics board while network connectivity is provided by a Ubiquiti Bullet M2HP mounted externally to one of the rear antenna masts on the vehicle. As a safety measure, a completely independent 2.4 GHz RC receiver is integrated into the system with the ability to override motor commands being sent from the Arduino microcontroller to the motor ESCs. More details on how these components are configured and their interactions are provided in the system diagram in figure 6.3.

In typical deployment scenarios, the ASV test platform was designed to work with a shore station consisting of a shore computer, Ubiquiti Rocket M2 WiFi base station, and a 2.4 GHz RC transmitter that serves as a failsafe override of the vehicle autonomy. The shore computer is primarily used to plan the coverage paths using the Constraint-Based Coverage Path Planners developed for this work and then relay these plans to the vehicle over WiFi. This communication is accomplished largely with the help of the ROS middleware, which is used across the software implementation of the field test system. After the planner determines a target coverage path, the path plan is packaged into custom Waypoint Command ROS messages that are then published to the network. The command interface on the shore computer additionally publishes Autonomy Command ROS messages that enable, disable, and control aspects of the test platform autonomy. The use of ROS in our implementation allows for the secondary role of the shore computer, which is to log all ROS messages published to the network whenever the platform is within range in case of catastrophic failure resulting in loss of the robot.

The software controlling the test vehicle is divided across a few ROS nodes implemented in Python running on the ODROID as well as some firmware written in C++ running on the Arduino Due. The firmware on the Arduino is written to communicate with the

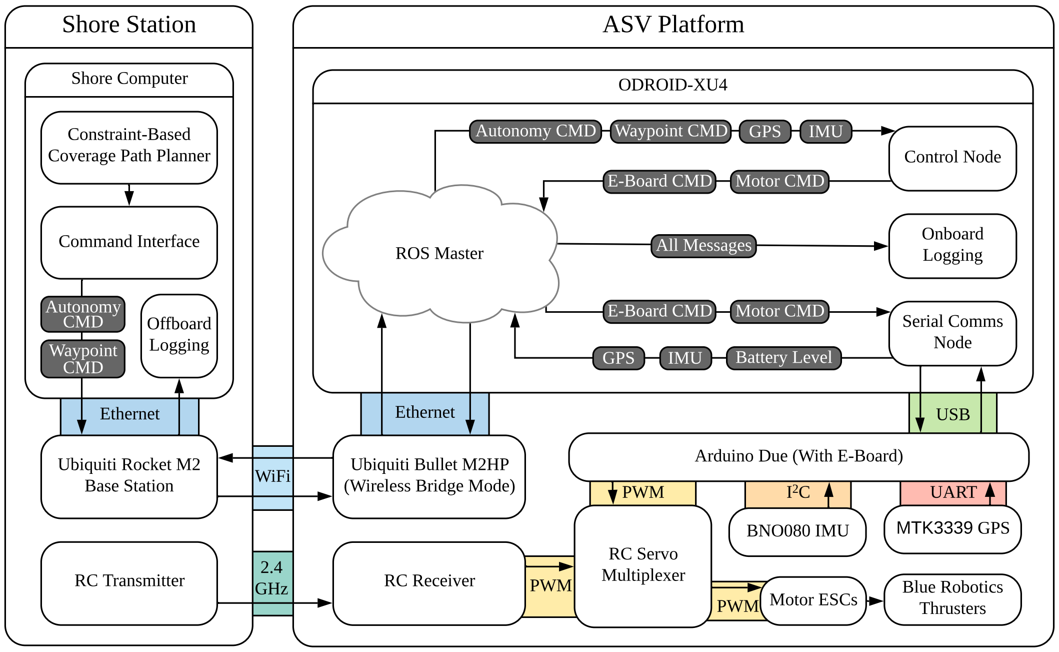


Figure 6.3: System diagram illustrating how the field testing platform functions

GPS sensor over a UART and an IMU over I^{2}C, package up the sensor readings as JSON messages, and send them up to the ODROID over USB. Futhermore, the firmware accepts motor commands as similarly formatted JSON messages from the ODROID, translates them into appropriate PWM commands, and sends them out to the appropriate motor ESC. On the ODROID side, the Serial Comms. Node translates the incoming sensor readings from JSON into the corresponding ROS messages and publishes them to the network. It also subscribes to any Motor Command ROS messages and upon receiving any, translates them to JSON motor commands and transmits it to the Arduino over USB. The ASV platform's autonomy is provided by the Control Node, which subscribes to all sensor ROS messages as well as Waypoint and Autonomy Commands from the shore station. Using this information, the node maintains a current path plan and computes appropriate motor commands to execute the path using a PID controller. It then publishes Motor Command messages that get passed to the Serial Comms. Node and out to the Arduino. All published ROS messages are recorded on the ODROID computer for later playback and analysis. Details of this architecture are illustrated in the system diagram shown in figure 6.3.

6.2 Results

In this section we present the results from our field experiments using the ASV test platform. We first describe the test site chosen for our experiments, then present the results of each experiment, and finally present some analysis to compare our results to simulation.

6.2.1 North Carolina - Cape Fear River

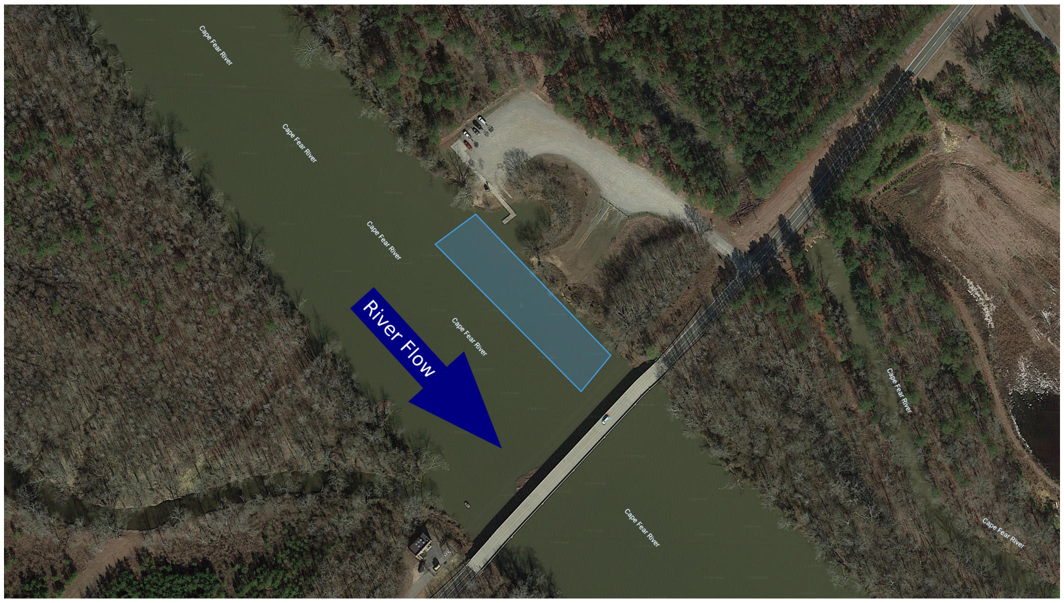
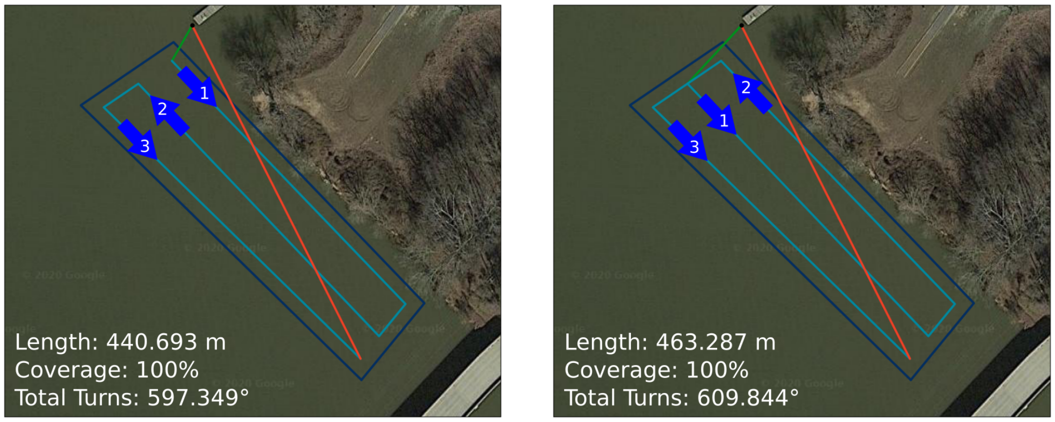


Figure 6.4: Test area on the Cape Fear River in North Carolina

For these experiments we selected a length of the Cape Fear River near a boat launch point, located just south of Raleigh, North Carolina. This testing location was selected due to its mild climate during the testing period, the significant variation in surface currents present near the bank, and the convenient access provided by the dock and boat ramp. The region highlighted in blue in figure 6.4 indicates the approximate boundary enclosing all experimental coverage paths tested at this site. The extent of this region into the river was limited due to the geometry of the river and its effect on the surface current distribution; while the significant depth variation near the bank resulted in substantial current variations near the shore, the surface currents towards the center of the river channel were more uniform as the depth leveled off. The direction of the river's flow is denoted with a blue arrow in 6.4.

River Section - Three Transects

In our first experiment we examine the effect of the energy-efficient sequencing optimization proposed in this work on the simplest scenario where the resulting coverage path can be differentiated from a conventional boustrophedon approach - a target region and payload footprint selection that requires three transects parallel to the shoreline to completely cover the area. If the experiment was limited to coverage paths with only two transects, every path produced by the energy-efficient optimization would be a valid boustrophedon path as well, and therefore would make for an inconclusive evaluation of the optimization's impact. The downstream length of the experimental test area was designed to limit the total length of the resulting coverage paths such that the test vehicle could execute each plan within approximately 15 minutes, depending on the river currents. This decision was made in order



(a) Boustrophedon Path

(b) Energy-Efficient Path

Figure 6.5: Three Transect Coverage Paths

to allow for multiple runs of each coverage pattern in a single day of testing as well as to limit the effect of changing ambient conditions during and between experimental runs.

In order to evaluate the impact of the energy-efficient optimization, two coverage plans were developed for the test area: a baseline boustrophedon path and an energy-efficient boustrophedon path with the optimization applied during planning. The planned coverage paths for this experiment are shown in figure 6.5, where the boundary of the test region is denoted in dark blue and each coverage path is drawn in light blue. A single point at the end of the dock was used as both the ingress and egress point for both coverage plans, and is illustrated by a black point on the maps. The ingress and egress routes that the test platform follows when starting and finishing execution of each coverage path are denoted in green and red respectively. Numbered blue arrows are overlaid over each transect in each coverage path to illustrate the direction of travel along each transect as well as their sequence of execution.

Comparing the two coverage plans, it is immediately apparent that the energy-efficient optimization has constrained the directions of the transects so that motion against the dominant surface currents occurs closer to the river bank, where the currents are slowest, and motion with the currents occurs along transects further out in the river channel, where the currents are faster. Since the scenario ingress point is located upstream of the test area, this strategy requires delaying execution of the transect closest to the riverbank, as opposed to the boustrophedon approach which greedily chooses to execute this transect first. As a result, the ingress route of the energy-efficient approach is slightly longer as the vehicle travels to the middle transect first. In addition, the distance travelled between the second and third transects is increased in the energy-efficient plan when compared to the baseline boustrophedon plan. Together these factors account for a 5.13% increase in the total path length over the baseline, which is in line with the additional path length introduced by the energy-efficient optimization in similar scenarios in simulation. The sharper angle of the ingress route with the middle transect on the energy-efficient route also results in a 2.09% increase in the total turns the vehicle makes during execution. Both coverage paths achieve complete coverage of the target region.

To collect the data for this experiment, the ASV test platform was deployed for a total of 16 trial runs, including 8 runs of the baseline boustrophedon plan and 8 runs of the energy-efficient plan. The runs were executed in an alternating fashion, with each run of the boustrophedon path being followed by a run of the energy-efficient path, in an effort to minimize the effects of changing river currents over time and to simplify analysis by ensuring each run of the energy-efficient path had a temporally adjacent run of the boustrophedon path to compare to. In between runs, the vehicle runtime, battery charge usage (Ah), and power consumption (Wh) were recorded and batteries were replaced if necessary. The net results from these runs are presented in table 6.1.

Coverage Path	Net Power Usage (Wh)	Net Current Usage (Ah)	Runtime
Boustrophedon	6.6	0.544	14:57.75
Energy-Efficient	5.7	0.472	12:49.35
Boustrophedon	6.4	0.528	14:39.48
Energy-Efficient	5.9	0.492	13:19.35
Boustrophedon	6.3	0.521	14:19.84
Energy-Efficient	5.5	0.457	12:26.00
Boustrophedon	6.1	0.507	13:47.32
Energy-Efficient	5.8	0.477	12:53.32
Boustrophedon	6.4	0.528	14:04.99
Energy-Efficient	5.6	0.465	12:31.05
Boustrophedon	6.7	0.553	14:51.57
Energy-Efficient	5.1	0.429	11:36.51
Boustrophedon	5.7	0.466	12:44.67
Energy-Efficient	5.3	0.440	11:48.42
Boustrophedon	5.9	0.480	12:50.94
Energy-Efficient	5.3	0.442	11:43.38
Boustrophedon Avg.	6.263	0.516	14:02.07
Energy-Efficient Avg.	5.525	0.459	12:23.42

Table 6.1: NC 3 Transect Coverage Path Energy Usage and Execution Times

An analysis of these results reveals that the energy-efficient path optimization successfully reduced the energy cost of covering the test region by approximately 0.7327 Watt-Hours on average, which amounts to a 11.78% improvement. A p-value of 3.53×10^{-4} from a two-sample heteroscedastic t-test using a two-tailed distribution suggests these results are significant. Furthermore, the average runtime of the optimized path is 1:38.65 faster than the baseline boustrophedon path, again representing approximately a 11.71% improvement.

When each set of trials runs is considered as a whole, some variation in the data is evident; the standard deviation of the energy cost, for instance, is 0.342 for the baseline boustrophedon path and 0.276 for the energy-efficient path. The recorded power consumption varies between 5.7 Wh and 6.7 Wh for the baseline plan, and between 5.1 Wh and 5.9 Wh for the energy-efficient plan, which correspond to ranges of 1.0 Wh and 0.8 Wh respectively. This substantial variation suggests that environmental conditions changed throughout the time-frame during which the experiment was conducted.

During data collection for this experiment, we observed the surface currents of the river changing gradually over the course of the day. Therefore, our hypothesis is that these

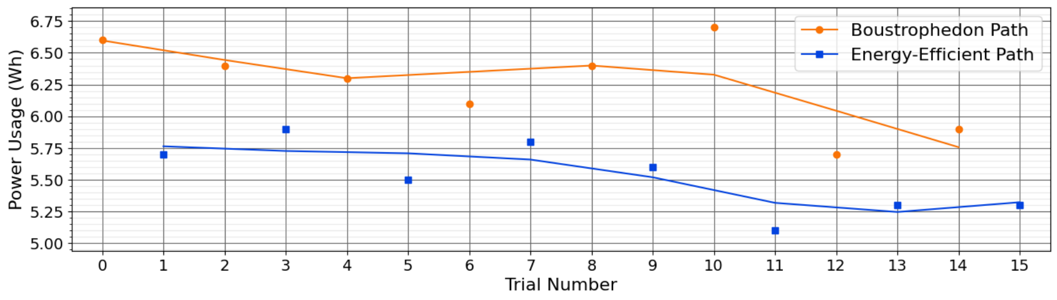


Figure 6.6: Energy usage for 3 Transect coverage paths

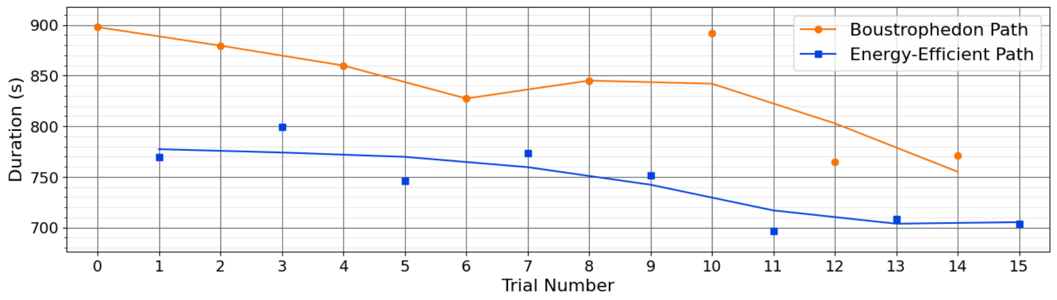


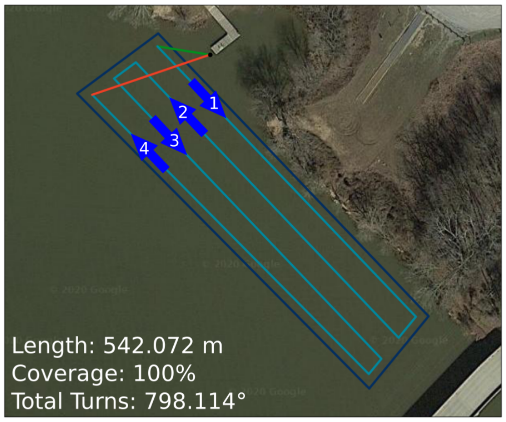
Figure 6.7: Execution times for 3 Transect coverage paths

current fluctuations may be responsible for the variation recorded across trial runs. To further investigate, we plot the power consumption recorded for each trial in the order they were executed to allow for a clearer comparison between temporally adjacent trial runs of the baseline boustrophedon and energy-efficient test plans. This representation is shown in figure 6.6, where the data corresponding to boustrophedon and energy-efficient trials are colored orange and blue respectively. Also shown in this figure are loess curves through each subset of the data, obtained through a local regression with a smoothing parameter of 0.5. From this analysis, we note that the loess curves for each type of coverage path seem to share a downwards trend over time and never intersect. A similar result is observed upon an equivalent analysis of the runtimes recorded for each trial, as shown in figure 6.7.

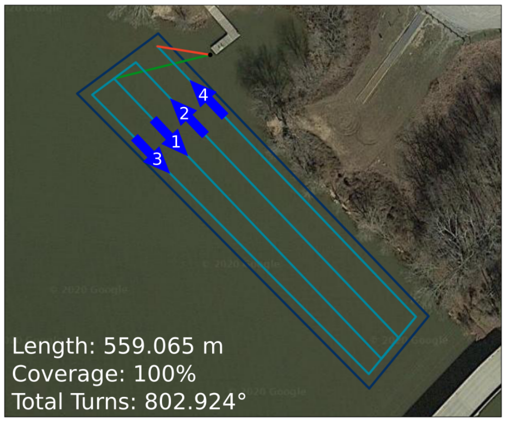
These results combined with our observations in the field seem to support our hypothesis that changes in environmental conditions did indeed occur over the course of this experiment and that these changes had a substantial impact on the execution of each coverage plan. The fact that the loess curves corresponding to the baseline boustrophedon data were strictly above the loess curves for the energy-efficient test data, however, lends further credence to our conclusion that the energy-efficient optimization did produce a significant reduction in power consumption over the baseline boustrophedon approach.

River Section - Four Transects

In this experiment we build upon the basic scenario designed for our first experiment, making several modifications to the target region and payload footprint specification that



(a) Boustrophedon Path



(b) Energy-Efficient Path

Figure 6.8: Four Transect Coverage Paths

we hypothesize will result in more significant energy savings from our energy-efficient optimization, based on our findings from simulation. For this test scenario, the target area is still confined to the section of river near the bank exhibiting substantial variations in surface currents, however, the length of this region parallel to the shoreline is extended to produce longer transects within each coverage path. The radius of the payload footprint is also decreased, in order to shift the two transects running along the borders of the test area towards the river bank and center, thereby increasing the differential between surface currents lying along those transects. This reduction in payload footprint has the added effect that the target region now requires four transects to cover completely with a boustrophedon pattern.

Repeating the previous experimental protocol, two coverage paths are generated once again for this revised test scenario: a baseline boustrophedon path and a modified boustrophedon path with our energy-efficient optimization applied. The planned coverage paths for this experiment are shown in figure 6.8 where the boundary of the test region is denoted in dark blue and each coverage path is drawn in light blue. The same point at the end of the dock used in the previous experiment was again selected as both the ingress and egress point for both coverage plans, and is illustrated by a black point on the maps. The ingress and egress routes that the test platform follows when starting and finishing execution of each coverage path are denoted in green and red respectively. Numbered blue arrows are overlaid over each transect in each coverage path to illustrate the direction of travel along each transect as well as their sequence of execution.

As in the previous experiment, the energy-efficient optimization has again refined the directions on the path constraints representing each transect in order to force all upstream travel to occur closer to the river bank. As a result, the two transects closest to the river bank in the energy-efficient path are restricted to upstream travel while the remaining transects further out in the river channel are limited to downstream travel, resulting in a path that spirals outwards from the center axis of the target region.

Although the ingress route for the energy-efficient path is longer than for the baseline boustrophedon path, the situation is reversed when it comes to each plan's egress route.

This results in only a marginal difference between the combined length of ingress and egress routes for each path. Furthermore, since both the baseline and energy-efficient approaches begin their first transect and finish their last transect at the upstream boundary of the target region, the ingress and egress routes make up a substantially smaller fraction of the complete coverage paths when compared to the previous experiment. This implies that the majority of the difference in length and energy consumption between the two coverage paths will be due to the influence of the energy-efficient optimization being tested. In total, the proposed energy-efficient path is 3.13% longer with 0.60% more turns than the baseline boustrophedon path. Both coverage paths achieve complete coverage of the target region.

To collect the data for this experiment, the ASV test platform was deployed for a total of 20 trial runs, including 10 runs of the baseline boustrophedon plan and 10 runs of the energy-efficient plan executed in an alternating fashion. In between runs, the vehicle runtime, battery charge usage (Ah), and power consumption (Wh) were recorded and batteries were replaced if necessary. The net results from these runs are presented in table 6.2.

Coverage Path	Net Power Usage (Wh)	Net Current Usage (Ah)	Runtime
Boustrophedon	7.6	0.628	18:17.81
Energy-Efficient	6.3	0.526	15:04.58
Boustrophedon	7.6	0.628	17:57.74
Energy-Efficient	6.9	0.577	16:32.40
Boustrophedon	7.8	0.644	18:42.03
Energy-Efficient	7.0	0.576	16:39.82
Boustrophedon	8.1	0.673	19:40.99
Energy-Efficient	7.1	0.588	17:10.64
Boustrophedon	8.0	0.659	19:15.85
Energy-Efficient	6.8	0.559	16:20.07
Boustrophedon	7.5	0.617	18:00.38
Energy-Efficient	6.7	0.553	16:14.88
Boustrophedon	8.4	0.690	19:49.74
Energy-Efficient	6.9	0.569	16:45.71
Boustrophedon	7.8	0.638	18:34.02
Energy-Efficient	6.5	0.540	15:40.93
Boustrophedon	7.6	0.627	18:10.21
Energy-Efficient	6.6	0.547	15:42.48
Boustrophedon	7.3	0.602	17:10.25
Energy-Efficient	6.2	0.521	14:52.32
Boustrophedon Avg.	7.77	0.641	18:33.90
Energy-Efficient Avg.	6.70	0.556	16:06.38

Table 6.2: NC 4 Transect Coverage Path Energy Usage and Execution Times

Upon analysis, these results suggest that the energy-efficient path optimization successfully reduced the energy cost of covering the test region by approximately 1.07 Watt-Hours on average, which amounts to a 13.77% improvement. A p-value of 4.44×10^{-7} from a two-sample heteroscedastic t-test using a two-tailed distribution suggests these results are significant. Furthermore, the average runtime of the optimized path is 2:27.52 faster than the baseline boustrophedon path, again representing approximately a 13.24% improvement.

Similarly to the previous experiment, there is also some variation in the data collected for each path type this time around; the standard deviation of the recorded energy cost is 0.3234 for the boustrophedon baseline path and 0.2981 energy-efficient path. The measured power consumption varies between 7.3 Wh and 8.4 Wh for the baseline plan, and between 6.2 Wh and 7.1 Wh for the energy-efficient plan, which correspond to ranges of 1.1 Wh and 0.9 Wh respectively. Although these values are in line with the statistics from the previous experiment, we note that the minimum energy cost across all runs of the baseline boustrophedon path is 7.3 Wh, which is still greater than the maximum energy cost across all energy-efficient trials, 7.1 Wh.

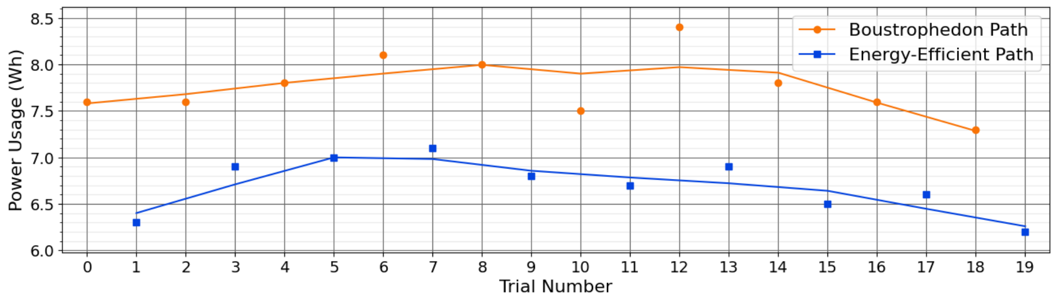


Figure 6.9: Energy usage for 4 Transect coverage paths

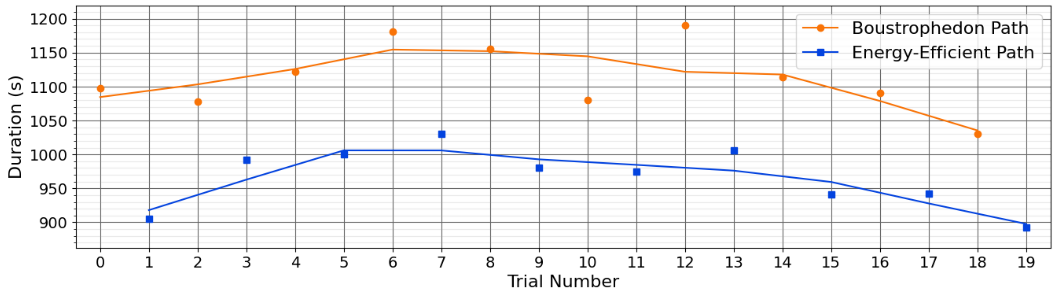


Figure 6.10: Execution times for 4 Transect coverage paths

In order to further verify these results, we performed the same temporal analysis as for the previous experiment, where each energy usage measurement is plotted in trial order to create an approximate time series. This representation is shown in figure 6.9, where the data corresponding to boustrophedon and energy-efficient trials are colored orange and blue respectively. Also shown in this figure are loess curves through each subset of the data, obtained through a local regression with a smoothing parameter of 0.5. A comparison of loess curves suggests that the data follow similar trends for both path types over the course of the experiment. A similar result is observed upon an equivalent analysis of the runtimes recorded for each trial, as shown in figure 6.10.

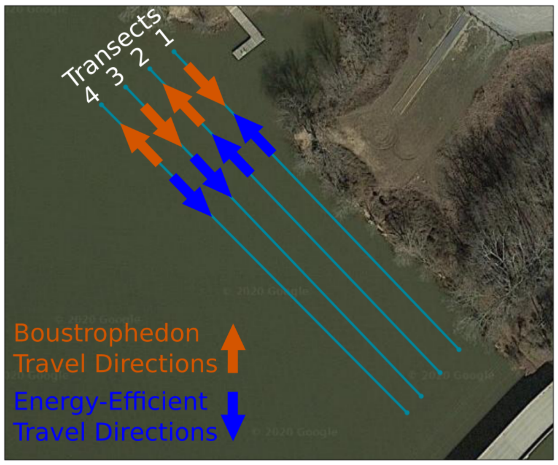
These results offer additional support to our hypothesis that the environmental conditions at this test location do change over time and influence the execution of each coverage plan. However, the fact that the loess curves maintain a steady separation and follow a similar trend is a strong indication that the reduction in power consumption produced by our energy-efficient optimization is a valid result despite changing ambient conditions. This coupled with the high statistical significance of the observed effect and the disjoint ranges of the power usage data recorded for each type of path, provides convincing evidence to support our claim that the energy-efficient optimization we have proposed does indeed reduce the energy costs of coverage paths for target regions with nonuniform current distributions.

6.3 Analysis

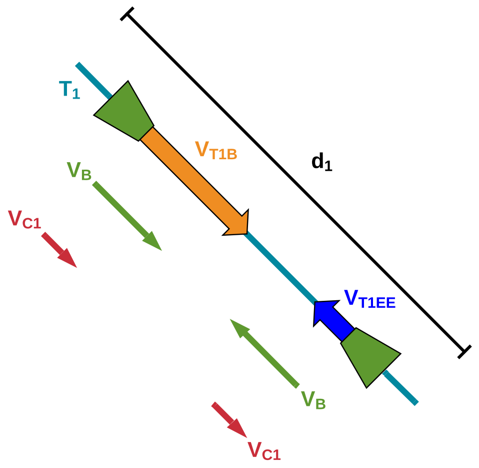
In the previous section we presented the results from field experiments that were designed to test the performance of the energy-efficient sequencing optimization proposed in this work. Although the data collected during these experiments was supportive of our hypothesis that the optimization would reduce the energy cost of coverage plans, we were interested in how the real world data would compare to simulated results. To this end, we developed an approach to use the experimental data to characterize the motion of the vehicle and the surface current distribution across the target region, and then use these models with our simulator to predict the execution times of our experimental paths.

In this section we walk through each step in our methodology and finally present the simulated coverage path durations alongside the measured values from the field experiments.

6.3.1 Estimating Vehicle Speed


Transects
4 3 2 1
Boustrophedon Travel Directions
Energy-Efficient Travel Directions

(a) Selecting transects with opposing execution



(b) Example of relevant speeds along transect 1

Figure 6.11: Using opposing transects to estimate boat speed

The first step in our analysis is to estimate the speed of the ASV test platform as it travels at a fixed thrust during execution of each coverage path. In order to accomplish this, we make a number of simplifying assumptions that facilitate extrapolating the average speed of the vehicle. First of all, we assume that the vehicle travels at a roughly constant speed, corresponding to its terminal velocity at a fixed thrust. Secondly, we assume that the transects in the experimental paths as well as the predominant direction of flow in the river are colinear. These are both reasonable simplifications as the experimental paths were designed to lie parallel to the direction of flow and are comprised of long transects, which reduce the turns required for coverage.

Trials	Speed Along T1 (m/s)	Speed Along T4 (m/s)	Average Speed (m/s)
0/1	0.6260	0.6336	0.6298
1/2	0.6534	0.6422	0.6478
2/3	0.6096	0.6607	0.6351
3/4	0.6185	0.6435	0.6310
4/5	0.6365	0.6449	0.6407
5/6	0.6667	0.6381	0.6524
6/7	0.6625	0.6204	0.6415
7/8	0.6216	0.6331	0.6273
8/9	0.6269	0.6516	0.6392
9/10	0.6431	0.6710	0.6571
10/11	0.6423	0.6514	0.6469
11/12	0.6411	0.6236	0.6324
12/13	0.6456	0.6288	0.6372
13/14	0.6374	0.6422	0.6398
14/15	0.6509	0.6507	0.6508
15/16	0.6253	0.6604	0.6428
16/17	0.6220	0.6543	0.6381
17/18	0.6327	0.6558	0.6443
18/19	0.6711	0.6441	0.6576
Avg.	0.6386	0.6448	0.6417
Std. Dev.	0.0170	0.0132	0.0088

Table 6.3: ASV Speed Estimates

The crux of our estimation methodology is to consider the transects for which the vehicle travels in opposite directions while executing the baseline boustrophedon and energy-efficient paths, as shown in figure 6.11a. For this scenario, we identify transects 1 and 4 as satisfying these conditions. To streamline our explanation we will focus our analysis on former, hereby referred to as $T1$, however the same technique can be used to estimate velocity using any transect with opposing travel directions.

The relevant values to this discussion are presented in figure 6.11b. We begin out analysis by determining the average speed of the vehicle motion along $T1$ in the world frame for both the boustrophedon and energy-efficient paths. These values, denoted as V_{T1B} and V_{T1EE} respectively, can be determined by dividing the length of the transect, d_1, by the duration of travel along $T1$ during execution of each path. We note that these velocity vectors are the effectively the superposition of the river current along that transect, $\overrightarrow{V_{C1}}$, and the terminal boat velocity vector, $\overrightarrow{V_B}$. This results in the following equations:

$$V_{T1B} = V_B + V_{C1} \qquad\qquad V_{T1EE} = V_B - V_{C1}$$

Adding these two equations together and solving for V_B, we find that the average boat speed along this transect can be determined by simply taking the average of the two transect speeds V_{T1B} and V_{T1EE}. The computed boat speeds along transects 1 and 4 for all pairs of consecutive trials are shown in table 6.3. We note the good agreement between boat speeds computed using data from $T1$ and $T4$, as well as the low standard deviation across all trials, which suggests the our methodology has determined a reasonable estimate for the terminal speed of the ASV and that the vehicle consistently achieved and maintained this speed throughout the experiments.

6.3.2 Estimating River Flow Along Transects

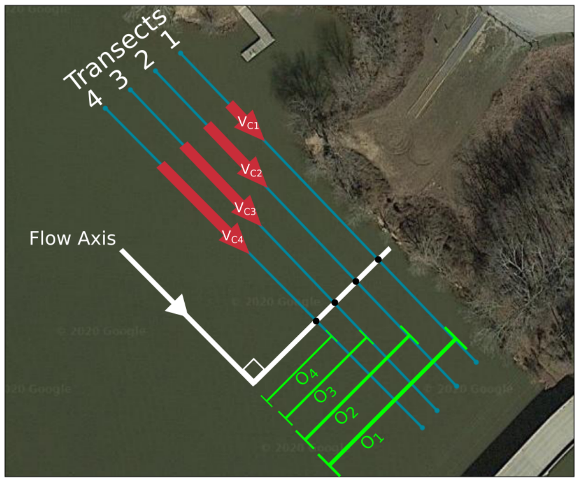


Figure 6.12: Flow estimated along each transect

Once we have an approximate speed for the ASV, we can use this information to estimate the average river current speed along each transect. Using the equations developed in the previous stage of our process, we solve for V_{Ci}, which represents the average flow speed along transect i, as shown by the red arrows in figure 6.12. In order to best account for any changes in environmental conditions or vehicle dynamics, we use a unique ASV speed value when computing the V_C values for each trial. Note that because the boat speed values determined in 6.3 all use data from two consecutive trials, each trial, with the exception of 0 and 19, have two potential speed values we can use when calculating flow velocities. To make use of all the data we have available, we take the average of these ASV speeds for trials 1 through 18. The vehicle speed used when computing flow speeds for each trial is listed in the second column of table 6.4.

Upon analysis of the flow estimates presented in table 6.4, we confirm the variation in surface currents that we observed while conducting our field experiments; the flow speeds in the river gradually increase over the course of the day before dropping off during the later experimental trials. Despite this trend, we note that the standard deviations for the flow speeds along each transect only range between 0.0192 and 0.0342 m/s.

6.3.3 River Current Model Reconstruction

In the final step of our process to prepare for the comparison of field experiment results with simulation, we use the estimated flow speeds along each transect to reconstruct a complete model of the surface currents throughout the test region. In order to achieve this, we set up a model of the river's surface currents that defines the flow at every point to lie along a specified flow axis vector. The magnitude of the flow defined by the model is a value determined by a quadratic function fit to the transect flow estimates we computed earlier

Trial	Boat Speed (m/s)	T1 Flow (m/s)	T2 Flow (m/s)	T3 Flow (m/s)	T4 Flow (m/s)
0	0.6298	0.0918	0.2574	0.3584	0.3500
1	0.6388	0.1084	0.2267	0.3313	0.3486
2	0.6415	0.1348	0.2602	0.3668	0.3444
3	0.6331	0.1902	0.2684	0.3754	0.3912
4	0.6359	0.1583	0.2560	0.3557	0.3732
5	0.6465	0.1676	0.3156	0.3450	0.3805
6	0.6469	0.2076	0.2932	0.3718	0.3979
7	0.6344	0.1639	0.3000	0.3935	0.3574
8	0.6333	0.1395	0.2947	0.3394	0.3589
9	0.6481	0.1672	0.2809	0.3348	0.3807
10	0.6520	0.1533	0.2860	0.3195	0.3388
11	0.6396	0.1602	0.2609	0.2641	0.3499
12	0.6348	0.1680	0.2928	0.3644	0.3771
13	0.6385	0.1501	0.2918	0.3276	0.3614
14	0.6453	0.1410	0.2738	0.3616	0.3607
15	0.6468	0.1313	0.2684	0.3396	0.3700
16	0.6405	0.0946	0.2539	0.3826	0.3365
17	0.6412	0.1324	0.2522	0.3754	0.3634
18	0.6509	0.1057	0.2451	0.3387	0.3439
19	0.6576	0.0722	0.2269	0.3095	0.3236
Avg.	0.6418	0.1419	0.2702	0.3478	0.3604
Std. Dev.	0.0073	0.0342	0.0240	0.0297	0.0192

Table 6.4: Flow Along Transect Estimates

and the perpendicular transect offsets from the river flow axis. The method of defining these offset distances with respect to the flow axis vector is illustrated in figure 6.12.

The process of fitting this flow magnitude function can be repeated for each set of flow speeds in table 6.4 to obtain a model of the river currents at the time of that particular trial. Alternatively, the average of these flow speeds can be used to generate a more general model. The vector field reconstruction resulting from such a generalized model of surface currents in the river is shown in figure 6.13. In the final segment of this section we describe the outcome of simulating execution of the boustrophedon and energy-efficient paths using these models and discuss how they compare to our field results.

6.3.4 Simulation

After we have successfully reconstructed the surface current model from our field data we can utilize this model with the simple simulator described in the previous chapter to simulate an ASV executing the boustrophedon and energy-efficient paths. We approached this analysis in two different ways: first using the generalized flow model built from the average ASV and transect flow speeds across all trials, and then using individualized models

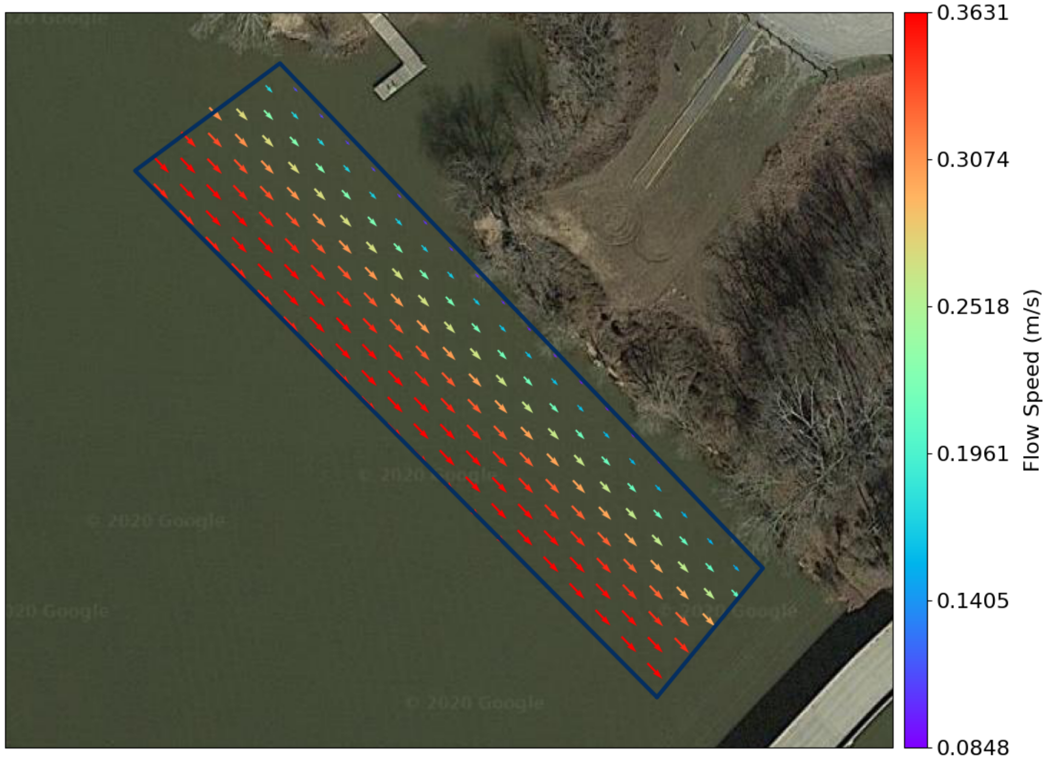


Figure 6.13: Reconstructed Surface Current Field

built from the data from each trial. The results of these simulations are presented alongside the experimental results from the field in figure 6.14.

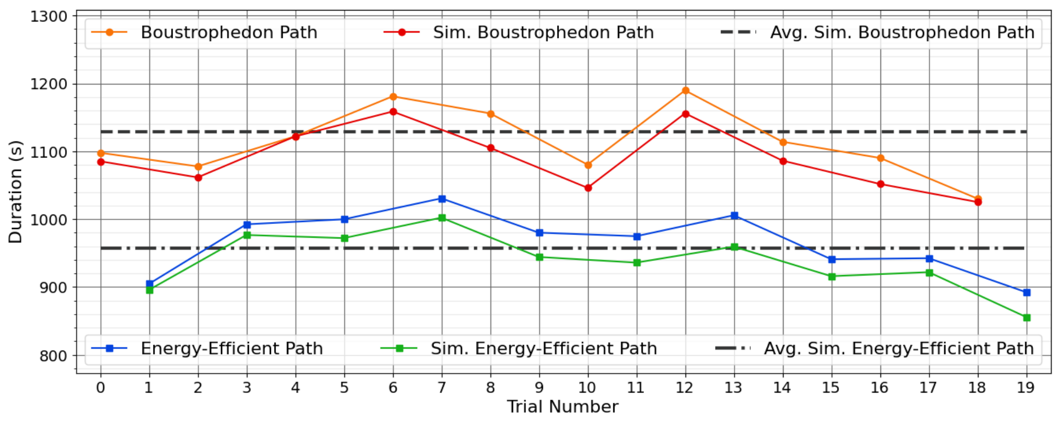


Figure 6.14: 4 Transect Path Execution Times - Field vs. Sim

In our generalized approach, we used the average transect flow speeds from table 6.4 to build a surface flow model and simulated execution of the boustrophedon and energy-energy efficient paths at a fixed ASV speed of 0.6418 m/s. This resulted in simulated execution times of 1129.15 and 956.90 seconds respectively, which corresponds to approximately a 15.25% improvement. This agrees well with the improvement we observed during our field experiments, where, on average, the energy-efficient path took 13.24% less time to execute than the boustrophedon path. These results are represented by the dashed and dot-dashed gray lines in figure 6.14.

As we have established when estimating the transect flow speeds from each set of trial data, the surface current distribution changed during the time the field experiments were conducted. As such, the recorded durations of each path type in the field vary below and above the values predicted by our generalized simulation. This effect is clearly illustrated by the plots of boustrophedon and energy-efficient trial durations, denoted in orange and blue respectively in figure 6.14. This motivated our second analysis approach where instead of using the general model with an average vehicle speed, we constructed a new model for each trial and simulated the path matching that trial at the true boat velocity calculated for that trial. The results from this analysis are shown as red and green lines in figure 6.14 and the values are recorded in table 6.5.

Trial	Simulated Duration (s)	Experimental Duration (s)
0	1085.29	1097.81
1	895.71	904.58
2	1061.54	1077.74
3	976.65	992.4
4	1121.88	1122.03
5	972.09	999.82
6	1158.6	1180.99
7	1002.17	1030.64
8	1105.11	1155.85
9	944.25	980.07
10	1046.17	1080.38
11	935.95	974.88
12	1155.94	1189.74
13	959.5	1005.71
14	1086.18	1114.02
15	915.95	940.93
16	1051.82	1090.21
17	921.92	942.48
18	1025.26	1030.25
19	855.76	892.32

Table 6.5: Simulated vs. Experimental Path Durations

Using this second approach, it is evident that the predicted execution times for both paths agree with the results recorded during our field experiments. In fact, when each simulated energy-efficient trial is compared against temporally adjacent simulated boustrophedon trials, the average reduction in execution time is 13.49%, which nearly matches the average improvement from our experimental results. An interesting pattern that emerges during comparison of the individually simulated and experimental trial runs, is that the simulation consistently underestimates the true execution time of each path, on average by 2 to 3 percent. This result is expected as the simple simulator we described in the previous chapter was designed with several simplifying assumptions cause it to underestimate the execution time for a path. Notably, the simulator assumes the vehicle can change velocity vectors instantaneously and in an optimal fashion. In reality the ASV cannot adjust its course instantaneously, must decelerate then accelerate when making turns, and has an imperfect controller that can produce some heading oscillations. These factors help explain the consistent difference between the simulated and experimental results.

6.4 Summary

In this chapter we introduced the autonomous surface vehicle that was developed specifically as a test platform for this work, presented experimental results from the field, and compared these results to simulation. Through our analysis we showed that the energy-efficient optimization presented in this work can significantly reduce the energy requirements for covering a given region in the presence of flow. Furthermore, despite river current variations across different trials, the energy-efficient path consistently delivered better performance than the baseline boustrophedon pattern. This suggests that our proposed optimization still produces energy savings when the exact magnitudes of the surface current are not known a priori; as long as the relative surface current distribution is generally accurate, the optimization can improve the energy-efficiency of the resulting coverage path. This is an important result because a complete and accurate knowledge of the surface currents in a river is seldom available and can change over the course of the day, as we noted in our experiments.

Chapter 7

Conclusions

Energy storage will continue to be a substantial obstacle for roboticists to grapple with for years to come; until battery technology catches up, energy-efficient algorithms will undoubtedly become an important piece of the toolset used to address this problem in the meantime. In this thesis we contribute to this toolset through the development of a new coverage path planning framework along with specific planning algorithms that enable more energy-efficient coverage of river domains. This is significant because to the best of our knowledge, this is the first result in robotics literature that addresses energy-efficient coverage planning in aquatic domains with moving fluids. Moreover, the CB-CPP framework developed provides a starting point towards a more generalized approach to building energy-efficient coverage planners from reusable components that can be combined and layered with each other to plan coverage paths to meet sophisticated performance criteria.

For completeness we conclude this discussion by briefly revisiting the expected contributions we outlined in chapter 1 and acknowledging where they were addressed within this thesis. The Constraint-Based Coverage Path Planning (CB-CPP) was introduced in chapter 3 and used to implement all coverage planners throughout this work. Chapter 4 details the development of specialized layout, refinement, and sequencing modules that can be combined with standard planning modules within the CB-CPP framework to plan more energy-efficient coverage paths through river domains. A set of new CPP planners using our proposed energy-efficient modules were developed over the course of chapters 3 and 4. These planners were tested in the simulation experiments presented in 5 and deployed on a custom built autonomous surface vehicle for real-world experiments that are described in chapter 6.

7.1 Future Work

We close out this thesis with a brief overview of promising avenues for further research into energy-efficient coverage of domains with moving fluids. Due to the substantial possibilities for extending the CB-CPP method to other domains and new applications, our discussion is organized into two sections: enhancements the methods proposed in this work, and system extensions that can expand the utility of this work to interesting new domains and applications.

7.1.1 Enhancements

During the development, implementation and testing of the methods proposed in this thesis, there were predictably times where certain features that would have greatly strengthened our proposed methodology or produced an added benefit, had to be omitted due to time and resource constraints. The following list presents the most poignant of these neglected enhancements:

- Implement thrust constraints in order to enable drifting with the currents when executing streamline constraints in the downstream direction. Thrust can be reduced to the point that control is still maintained in order to allow the vehicle to make course corrections and stay on the constraint as it drifts.

- Allow for coverage overlap bias optimization to bias paths outside of the payload configuration space (e.g., if vehicle radius is smaller than sensor radius, we can push the innermost path constraints closer to the domain boundary to shorten them further).

7.1.2 System Extensions

As the CB-CPP framework was developed and used to create new coverage plans that achieved interesting and useful objectives, numerous natural extensions continuously became evident. The following list captures a sampling of these ideas:

- Constraints on a variety of different vehicle parameters (e.g., velocity constraints to effectively enforce speed limits through dangerous sectors for safety).

- Actuator/Sensing shutdown when moving through regions that have already been covered (e.g., shut off/decrease rate of vacuum/sensors while moving between constraints or covered cells).

- Turn off nonessential equipment on certain constraints or portions of constraints (e.g., if we know the approximate communications range, shut down wireless communications for all constraint segments outside this coverage area).

- Consider wind fields in addition to current fields for river domains.

- Extend energy-efficient coverage planning techniques to planning for UAV coverage in the presence of winds (e.g., planning bridge inspection routes while utilizing the bridge structure to avoid winds).

- Extend energy-efficient coverage planning techniques to planning for ground vehicle coverage over uneven terrain.

Bibliography

[1] E U Acar and H Choset. Robust sensor-based coverage of unstructured environments. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 1, pages 61–68, 2001. 11

[2] Ercan U Acar and Howie Choset. Sensor-based Coverage of Unknown Environments: Incremental Construction of Morse Decompositions. *The International Journal of Robotics Research*, 21(4):345–366, 4 2002. 11

[3] Ercan U Acar, Howie Choset, Alfred A Rizzi, Prasad N Atkar, and Douglas Hull. Morse Decompositions for Coverage Tasks. *I. J. Robotics Res.*, 21:331–344, 2002. 11

[4] D S Apostolopoulos, L Pedersen, B N Shamah, K Shillcutt, M D Wagner, and W L Whittaker. Robotic Antarctic meteorite search: outcomes. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 4, pages 4174–4179, 2001. 19

[5] Dimitrios S Apostolopoulos, Michael D Wagner, Benjamin N Shamah, Liam Pedersen, Kimberly Shillcutt, and William L Whittaker. Technology and Field Demonstration of Robotic Search for Antarctic Meteorites. *The International Journal of Robotics Research*, 19(11):1015–1032, 11 2000. 19

[6] Esther M Arkin, Sándor P Fekete, and Joseph S B Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50, 2000. 5

[7] Esther M Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994. 5

[8] M Bertoldi, M Yardimci, C M Pistor, and S I Guceri. Domain Decomposition and Space Filling Curves in Toolpath Planning and Generation, 1998. 14

[9] R Bormann, F Jordan, J Hampp, and M Hägele. Indoor Coverage Path Planning: Survey, Implementation, Analysis. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1718–1725, 2018. 5

[10] R Bormann, F Jordan, W Li, J Hampp, and M Hägele. Room segmentation: Survey, implementation, and analysis. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1019–1026, 2016. 13

[11] Zuo Llang Cao, Yuyu Huang, and Ernest L. Hall. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic Systems*, 1988. 5

[12] J A Castellanos, J D Tardos, and G Schmidt. Building a global map of the environment of a mobile robot: the importance of correlations. In *Proceedings of International Conference on Robotics and Automation*, volume 2, pages 1053–1059, 1997. 10

[13] Y Choi, T Lee, S Baek, and S Oh. Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5788–5793, 2009. 9

[14] Howie Choset. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition. *Autonomous Robots*, 9:247–253, 2000. 11, 28

[15] Howie Choset. Coverage for robotics - A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 2001. 5, 6

[16] Howie Choset and Philippe Pignon. Coverage Path Planning: The Boustrophedon Decomposition. In *International Conference on Field and Service Robotics*, 1 1997. 11

[17] Pai Chou and G Borriello. Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems. In *31st Design Automation Conference*, pages 1–4, 1994. 15

[18] Jason Derenick, Nathan Michael, and Vijay Kumar. Energy-aware coverage control with docking for robot teams. In *IEEE International Conference on Intelligent Robots and Systems*, 2011. 17

[19] Tamal K Dey and Wulue Zhao. Approximating the Medial Axis from the Voronoi Diagram with a Convergence Guarantee. *Algorithmica*, 38(1):179–200, 2004. 43

[20] Carmelo Di Franco and Giorgio Buttazzo. Coverage Path Planning for UAVs Photogrammetry with Energy and Resolution Constraints. *Journal of Intelligent & Robotic Systems*, 83(3):445–462, 2016. 48

[21] William E Dietrich, J Dungan Smith, and Thomas Dunne. Flow and Sediment Transport in a Sand Bedded Meander. *The Journal of Geology*, 87(3):305–315, 1979. 62

[22] Donghong Ding, Zengxi (Stephen) Pan, Dominic Cuiuri, and Huijun Li. A tool-path generation strategy for wire and arc additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 73(1):173–183, 2014. 12

[23] S Dogru and L Marques. Towards fully autonomous energy efficient Coverage Path Planning for autonomous mobile robots on 3D terrain. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015. 19

[24] Sedat Dogru and Lino Marques. Energy Efficient Coverage Path Planning for Autonomous Mobile Robots on 3D Terrain. *International Conference on Autonomous Robot Systems and Competitions*, pages 118–123, 2015. 19

[25] Tawfik T El-Midany, Ahmed Elkeran, and Hamdy Tawfik. Toolpath Pattern Comparison: Contour-Parallel with Direction-Parallel. *Geometric Modeling and Imaging–New Trends (GMAI'06)*, pages 77–82, 2006. 14

[26] A Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987. 7

[27] Y Gabriely and E Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1927–1933, 2001. 8

[28] Y Gabriely and E Rimon. Spiral-STC: an on-line coverage algorithm of grid environments by a mobile robot. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 954–960, 2002. 8

[29] Douglas W Gage. Randomized search strategies with imperfect sensors. In *Proc.SPIE*, volume 2058, 2 1994. 14

[30] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 2013. 5, 7

[31] P M Gerhart, A L Gerhart, and J I Hochstein. *Munson, Young and Okiishi's Fundamentals of Fluid Mechanics*. Wiley, 8 edition, 2016. 60

[32] Ian Gibson, David W Rosen, and Brent Stucker. *Additive Manufacturing Technologies: Rapid Prototyping to Direct Digital Manufacturing*. Springer Publishing Company, Incorporated, 1st edition, 2009. 6, 12, 14

[33] E Gonzalez, M Alarcon, P Aristizabal, and C Parra. BSA: a coverage algorithm. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1679–1684, 2003. 9

[34] E Gonzalez, O Alvarez, Y Diaz, C Parra, and C Bustacara. BSA: A Complete Coverage Algorithm. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2040–2044, 2005. 9

[35] Martin Held. On the Computational Geometry of Pocket Machining. In *Lecture Notes in Computer Science*, 1991. 6, 12, 14

[36] Susan Hert, Sanjay Tiwari, and Vladimir Lumelsky. A terrain-covering algorithm for an AUV. *Autonomous Robots*, 3(2):91–119, 1996. 6

[37] A L HODGKIN and A F HUXLEY. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 8 1952. 9

[38] W H Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 1, pages 27–32, 2001. 12

[39] R.A. Jarvis and J.C. Byrne. Robot Navigation: Touching, Seeing and Knowing. In *Proceedings of 1st Australian Conference on Artificial Intelligence*, 1986. 8

[40] P A Jimenez, Bijan Shirinzadeh, A Nicholson, and Gursel Alici. Optimal area covering using genetic algorithms. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–5, 2007. 13

[41] G Q Jin, W D Li, and L Gao. An adaptive process planning approach of rapid prototyping and manufacturing. *Robotics and Computer-Integrated Manufacturing*, 29(1):23–38, 2013. 14

[42] Jian Jin and Lie Tang. Optimal Coverage Path Planning for Arable Farming on 2D Surfaces. *Transactions of the ASABE*, 53:283–295, 2010. 18

[43] Jian Jin and Lie Tang. Coverage path planning on three-dimensional terrain for arable farming. *Journal of Field Robotics*, 28(3):424–440, 5 2011. 19

[44] Yu-an Jin, Yong He, Jian-zhong Fu, Wen-feng Gan, and Zhi-wei Lin. Optimization of tool-path generation for material extrusion-based additive manufacturing technology. *Additive Manufacturing*, 1-4:32–47, 2014. 6

[45] D Jones and G A Hollinger. Planning Energy-Efficient Trajectories in Strong Disturbances. *IEEE Robotics and Automation Letters*, 2(4):2080–2087, 2017. 17

[46] M Kalakrishnan, S Chitta, E Theodorou, P Pastor, and S Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011. 17

[47] Farid Karimipour, Mehran Ghandehari, and Hugo Ledoux. Medial Axis Approximation of River Networks for Catchment Area Delineation. In Alias Abdul Rahman, Pawel Boguslawski, Christopher Gold, and Mohamad Nor Said, editors, *Developments in Multidimensional Spatial Data Models*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. 36, 43

[48] Chong Hui Kim and Byung Kook Kim. Minimum-energy translational trajectory generation for differential-driven wheeled mobile Robots. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2007. 17

[49] David C Ku and Giovanni De Micheli. Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):696–718, 1992. 15

[50] Simon Lacroix, Anthony Mallet, David Bonnafous, Gérard Bauzil, Sara Fleury, Matthieu Herrb, and Raja Chatila. Autonomous Rover Navigation on Unknown Terrains: Functions and Integration. *The International Journal of Robotics Research*, 21(10-11):917–942, 10 2002. 15

[51] Jean-Claude Latombe. Exact Cell Decomposition. In Jean-Claude Latombe, editor, *Robot Motion Planning*, pages 200–247. Springer US, Boston, MA, 1991. 10, 28

[52] S L Laubach and J W Burdick. An autonomous sensor-based path-planner for planetary microrovers. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 347–354, 1999. 15

[53] Nicholas R J Lawrance and Salah Sukkarieh. Path planning for autonomous soaring flight in dynamic wind fields. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2011. 17

[54] T Lee, S Baek, S Oh, and Y Choi. Complete coverage algorithm based on linked smooth spiral paths for mobile robots. In *2010 11th International Conference on Control Automation Robotics & Vision*, pages 609–614, 2010. 9

[55] Luna Bergere Leopold and M Gordon Wolman. River meanders. *Geological Society of America Bulletin*, 71(6):769–793, 1960. 48, 62

[56] Elzbieta Lewandowicz and Paweł Flisek. A method for generating the centerline of an elongated polygon on the example of a watercourse. *ISPRS International Journal of Geo-Information*, 9(5), 5 2020. 36, 43

[57] Jinfeng Liu, Pai H Chou, Nader Bagherzadeh, and Fadi Kurdahi. Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems. 15

[58] Yu Liu, Xiaoyong Lin, and Shiqiang Zhu. Combined coverage path planning for autonomous cleaning robots in unstructured environments. In *2008 7th World Congress on Intelligent Control and Automation*, pages 8271–8276, 2008. 14

[59] T Lolla, M P Ueckermann, K Yi, P J Haley, and P F J Lermusiaux. Path Planning in Time Dependent Flow Fields using Level Set Methods. 16

[60] Chaomin Luo, S X Yang, D A Stacey, and J C Jofriet. A solution to vicinity problem of obstacles in complete coverage path planning. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 612–617, 2002. 9

[61] Tsugio Makimoto and Yoshio Sakai. Evolution of Low Power Electronics and Its Future Applications. 1970. 15

[62] Takashi Matsuyama and Tsai Yun Phillips. Extracting the Medial Axis from the Voronoi Diagram of Boundary Segments: An Alternative Method for Closed Boundary Detection. Technical report, 1983. 43

[63] Michael McAllister and Jack Snoeyink. Medial Axis Generalization of River Networks. *Cartography and Geographic Information Science*, 27(2):129–138, 1 2000. 36, 43

[64] Yongguo Mei. *Energy-Efficient Mobile Robots*. PhD thesis, Purdue University, 2007. 15

[65] Yongguo Mei, Yung-Hsiang Lu, Y C Hu, and C S G Lee. A case study of mobile robot's energy consumption and conservation techniques. In *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pages 492–497, 2005. 15

[66] Yongguo Mei, Yung Hsiang Lu, C. S George Lee, and Y. Charlie Hu. Energy-efficient mobile robot exploration. *Proceedings - IEEE International Conference on Robotics and Automation*, 2006(May):505–511, 2006. 15

[67] S Michaud, A Schneider, R Bertrand, P Lamon, R Siegwart, M Van Winnendael, and A Schiele. SOLERO: SOLAR-POWERED EXPLORATION ROVER. 15

[68] J Milnor, M SPIVAK, and R WELLS. *Morse Theory. (AM-51), Volume 51*. Princeton University Press, 1969. 11

[69] Stewart Moorehead, Reid Simmons, Dimitrios (Dimi) Apostolopoulos, and William (Red) L Whittaker. Autonomous Navigation Field Results of a Planetary Analog Robot in Antarctica. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 6 1999. 15

[70] H Moravec and A Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, 1985. 7

[71] Joon Seop Oh, Yoon Ho Choi, Jin Bae Park, and Y F Zheng. Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, 51(3):718–726, 2004. 7

[72] Timo Oksanen and Arto Visala. Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 2009. 11, 12, 13, 18

[73] J Palacin, T Palleja, I Valganon, R Pernia, and J Roca. Measuring Coverage Performances of a Floor Cleaning Mobile Robot Using a Vision System. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4236–4241, 2005. 13

[74] Christos H Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. 5

[75] L Paull, S Saeedi, H Li, and V Myers. An information gain based adaptive path planning method for an autonomous underwater vehicle using sidescan sonar. In *2010 IEEE International Conference on Automation Science and Engineering*, pages 835–840, 2010. 7

[76] L Paull, S Saeedi, M Seto, and H Li. Sensor-Driven Online Coverage Planning for Autonomous Underwater Vehicles. *IEEE/ASME Transactions on Mechatronics*, 18(6):1827–1838, 2013. 7

[77] Franco P Preparata and Michael I Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, Berlin, Heidelberg, 1985. 10

[78] F. B. Prinz and Jyh-Huei Chern. Geometric abstractions using medial axis transformation. *Carnegie Mellon, Engineering Design Research Center*, 1 1988. 36, 43

[79] Xuena Qiu, Jiatao Song, Xuejun Zhang, and Shirong Liu. A Complete Coverage Path Planning Method for Mobile Robot in Uncertain Environments. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 8892–8896, 2006. 10

[80] Rajesh Ramamurthy and Rida T Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries I. Theoretical foundations. *Journal of Computational and Applied Mathematics*, 102(1):119–141, 1999. 43

[81] Laura E Ray, James H Lever, Alexander D Streeter, and Alexander D Price. Design and Power Management of a Solar-Powered " Cool Robot " for Polar Instrument Networks. *Journal of Field Robotics*, 24(7):581–599, 2007. 15

[82] John Reif and Zheng Sun. An Efficient Approximation Algorithm for Weighted Region Shortest Path Problem. 10 1999. 16

[83] C Sauzé and M Neal. Artificial Endocrine Controller for Power Management in Robotic Systems. *IEEE Transactions on Neural Networks and Learning Systems*, 24(12):1973–1985, 2013. 17

[84] Colin Sauze and Mark Neal. Long term power management in sailing robots. In *OCEANS 2011 IEEE - Spain*, 2011. 17

[85] Paul Scerri, Prasanna Velagapudi, Balajee Kannan, Abhinav Valada, Christopher Tomaszewski, John M Dolan, Adrian Scerri, Kumar Shaurya Shankar, Luis Lorenzo Bill-Clark, and George A Kantor. Real-World Testing of a Multi-Robot Team. In Padgham Conitzer Winikoff and van der Hoek, editors, *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, www.ifaamas.org, 6 2012. International Foundation for Autonomous Agents and Multiagent Systems. 16, 18

[86] Homayoun Seraji and Ayanna Howard. Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation*, 2002. 15

[87] K Shillcutt and W Whittaker. Solar navigational planning for robotic explorers. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1421–1426, 2001. 19

[88] Kimberly J Shillcutt. *Solar Based Navigation for Robotic Explorers*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2000. 19

[89] V Shivashankar, R Jain, U Kuter, and Dana S Nau. Real-Time Planning for Covering an Initially-Unknown Spatial Environment. 8 2011. 8

[90] A Stentz and M Hebert. A complete navigation system for goal acquisition in unknown environments. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 425–432, 1995. 15

[91] Grant P. Strimel and Manuela M. Veloso. Coverage planning with finite resources. In *IEEE International Conference on Intelligent Robots and Systems*, 2014. 18

[92] Zheng Sun and John Reif. BUSHWHACK: An approximation algorithm for minimal paths through pseudo-Euclidean spaces. *Proceedings of the 12th Annual International Symposium on Algorithms and Computation*, 2223, 10 2001. 16

[93] Zheng Sun and John H. Reif. On finding energy-minimizing paths on terrains. *IEEE Transactions on Robotics*, 2005. 16

[94] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998. 10

[95] Sebastian Thrun. Robotic Mapping: A Survey. In *Exploring Artificial Intelligence in the New Millennium*, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. 10

[96] Pratap Tokekar, Nikhil Karnad, and Volkan Isler. Energy Optimal Velocity Profiles for Car-like Robots. 2011. 16

[97] Pratap Tokekar, Nikhil Karnad, and Volkan Isler. Energy-Optimal Velocity Profiles for Car-Like Robots. In *2011 IEEE International Conference on Robotics and Automation*, pages 1457–1462, 2011. 16

[98] C. Tomaszewski, A. Valada, and P. Scerri. Planning efficient paths through dynamic flow fields in real world domains. In *OCEANS 2013 MTS/IEEE - San Diego: An Ocean in Common*, 2013. 16

[99] P Tompkins, A Stentz, and D Wettergreen. Global path planning for Mars rover exploration. In *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, volume 2, pages 801–815, 2004. 16

[100] P Tompkins, A Stentz, and W Whittaker. Mission planning for the Sun-Synchronous Navigation Field Experiment. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 4, pages 3493–3500, 2002. 16

[101] Paul Tompkins, Anthony Stentz, and David Wettergreen. Mission-level path planning and re-planning for rover exploration. In *Robotics and Autonomous Systems*, 2006. 16

[102] E Tunstel, T Huntsberger, H Aghazarian, P Backes, E Baumgartner, Yang Cheng, M Garrett, B Kennedy, C Leger, L Magnone, J Norris, M Powell, A Trebi-Ollennu, and P Schenker. FIDO rover field trials as rehearsal for the NASA 2003 Mars Exploration Rovers mission. In *Proceedings of the 5th Biannual World Automation Congress*, volume 14, pages 320–327, 2002. 15

[103] Abhinav Valada, Prasanna Velagapudi, Balajee Kannan, Christopher Tomaszewski, George A Kantor, and Paul Scerri. Development of a Low Cost Multi-Robot Autonomous Marine Surface Platform. In *The 8th International Conference on Field and Service Robotics (FSR 2012)*, 7 2012. 16, 18

[104] J. R. VanderHeide and N. S. V. Rao. Terrain coverage of an unknown room by an autonomous mobile robot. Technical report, Oak Ridge National Lab, Oak Ridge, TN, 1995. 11

[105] Soumya Vasisht, Mehran Mesbahi, and William E Boeing. Trajectory Design and Coverage Control for Solar-Powered UAVs. 19

[106] Guiling Wang, Mary Jane Irwin, Haoying Fu, Piotr Berman, Wensheng Zhang, and Tom La Porta. Optimizing sensor movement planning for energy efficiency. *ACM Transactions on Sensor Networks*, 2011. 16

[107] T Wasser, A Jayal, and C Pistor. Implementation and Evaluation of Novel Buildstyles in Fused Deposition Modeling (FDM). 1999. 14

[108] David Wettergreen, Nathalie Cabrol, Vijayakumar Baskaran, Francisco Calderon, Stuart Heys, Dominic Jonak, Rolf Allan Luders, David Pane, Trey Smith, James Teza, Paul Tompkins, Daniel Villa, Chris Williams, and Michael D Wagner. Second Experiments in the Robotic Investigation of Life in the Atacama Desert of Chile. In *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Pittsburgh, PA, 9 2005. Robotics Institute , Carnegie Mellon University. 16

[109] David Wettergreen, M Bernardine Dias, Benjamin Shamah, James Teza, Paul Tompkins, Christopher Urmson, Michael D Wagner, and William (Red) L Whittaker. First Experiment in Sun-Synchronous Exploration. In *International Conference on Robotics and Automation*, pages 3501–3507, Pittsburgh, PA, 5 2002. Robotics Institute , Carnegie Mellon University. 16

[110] David Wettergreen, Benjamin Shamah, Paul Tompkins, and William (Red) L Whittaker. Robotic Planetary Exploration by Sun-Synchronous Navigation. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS '01)*, Pittsburgh, PA, 6 2001. Robotics Institute , Carnegie Mellon University. 16

[111] S C Wong and B A MacDonald. A topological coverage algorithm for mobile robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1685–1690, 2003. 11

[112] Sylvia Wong. *Qualitative Topological Coverage of Unknown Environments by Mobile Robots*. PhD thesis, University of Auckland, 2 2006. 11

[113] Sylvia C Wong and Bruce A MacDonald. Complete Coverage by Mobile Robots Using Slice Decomposition Based on Natural Landmarks. In Chengqi Zhang, Hans W. Guesgen, and Wai-Kiang Yeap, editors, *PRICAI 2004: Trends in Artificial Intelligence*, pages 683–692, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 11, 28

[114] C Wu, C Dai, X Gong, Y Liu, J Wang, X D Gu, and C C L Wang. Energy-Efficient Coverage Path Planning for General Terrain Surfaces. *IEEE Robotics and Automation Letters*, 4(3):2584–2591, 2019. 6, 19

[115] S X Yang and C Luo. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):718–724, 2004. 9

[116] Y Yang, H T Loh, J Y H Fuh, and Y G Wang. Equidistant path generation for improving scanning efficiency in layered manufacturing. *Rapid Prototyping Journal*, 8:30–37, 3 2002. 14

[117] A Zelinsky, R A Jarvis, J C Byrne, and S Yuta. Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In *In Proceedings of International Conference on Advanced Robotics*, pages 533–538, 1993. 8

[118] Alexander Zelinsky. *Environment exploration and path planning algorithms for mobile robot navigation using sonar*. PhD thesis, University of Wollongong, 1991. 8

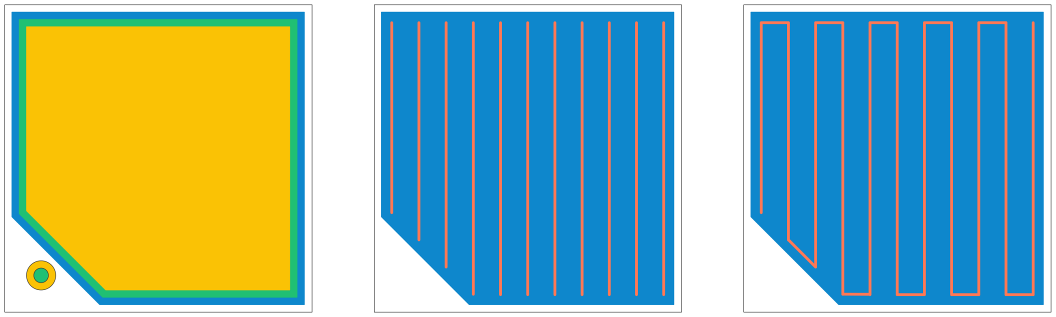
[119] Haisen Zhao, Fanglin Gu, Qi-Xing Huang, Jorge Garcia, Yong Chen, Changhe Tu, Bedrich Benes, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Connected Fermat Spirals for Layered Fabrication. *ACM Trans. Graph.*, 35(4), 7 2016. 6, 7, 14, 19

Appendix A

Constraint-Based Coverage Planner Output Examples

In this appendix we provide additional examples of coverage planners implemented within the CB-CPP framework. Throughout the appendix we include tables that show coverage planner output as different parameters of the planning problem are changed individually, in order to illustrate the effect changing these parameters may have on the final coverage path. Furthermore, we also include a table of different coverage plans for a variety of scenarios to demonstrate the robustness of the planners developed using the CB-CPP framework.

The most common types of graphics included within this appendix are shown below in figure A.1. Within this figure, image A.1a shows the domain for the planning scenario in blue, as well as the vehicle and payload footprints in green and yellow respectively. The configuration spaces associated with these footprints are shown in their respective colors overlaid over the domain polygon. Image A.1b illustrates the path constraints as they are laid out by the planner, while image A.1c shows the final coverage path produced for the region.



(a) Domain and Vehicle Config. (b) Path Constraints from Planner (c) Final Coverage Path

Figure A.1: An example of the figures showing planner input and output

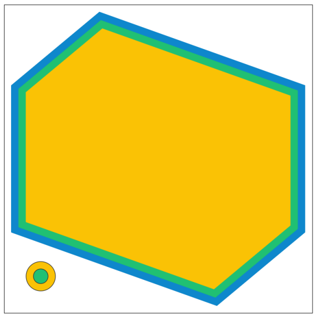
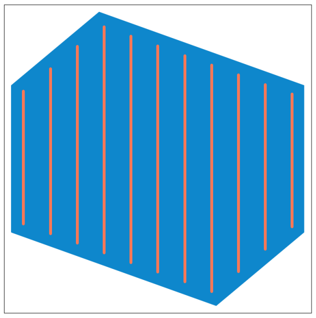
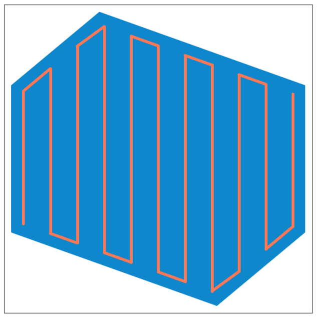
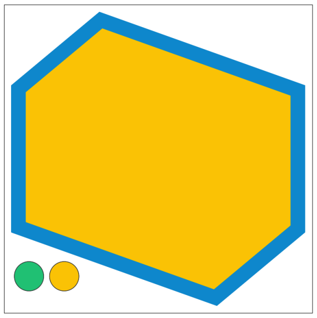
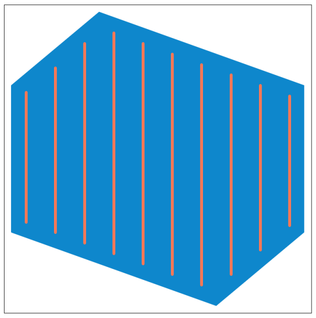
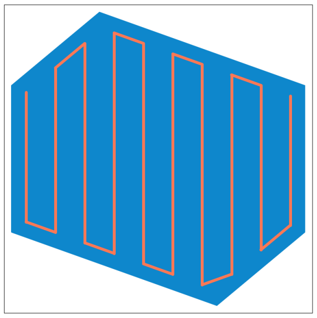
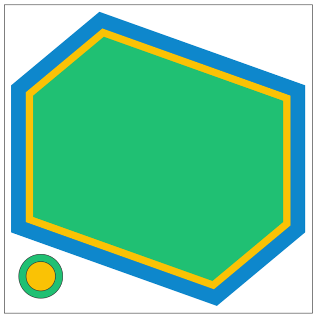
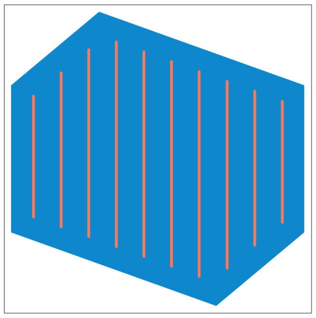
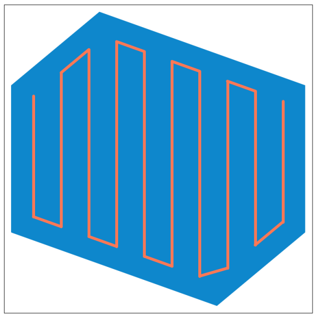
Domain & Vehicle Config.	Constraint Layout	Coverage Path
 Vehicle Radius: 2.5 m Payload Radius: 5.0 m	 Num. of Constraints: 11 Coverage Rate: 99.22%	 Length: 827.12 m Total Turns: 1800.00°
 Vehicle Radius: 5.0 m Payload Radius: 5.0 m	 Num. of Constraints: 10 Coverage Rate: 97.32%	 Length: 736.99 m Total Turns: 1620.00°
 Vehicle Radius: 7.5 m Payload Radius: 5.0 m	 Num. of Constraints: 10 Coverage Rate: 87.18%	 Length: 688.41 m Total Turns: 1620.00°

Table A.1: Boustrophedon Coverage Plans with Changing Vehicle Envelope

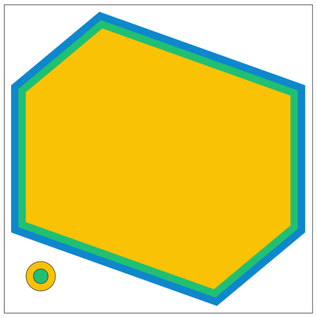
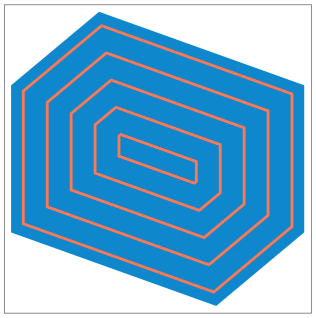
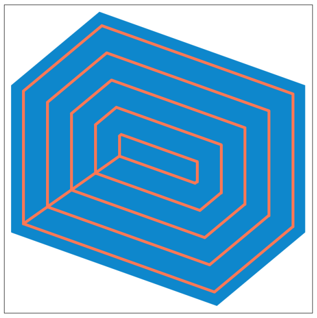
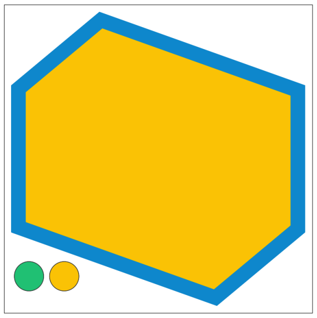
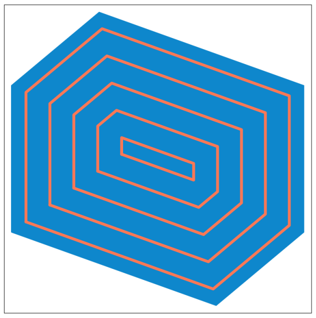
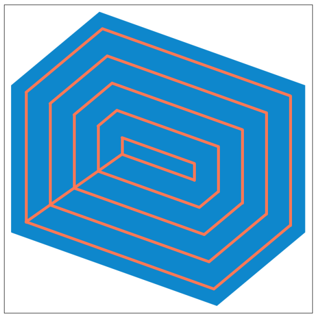
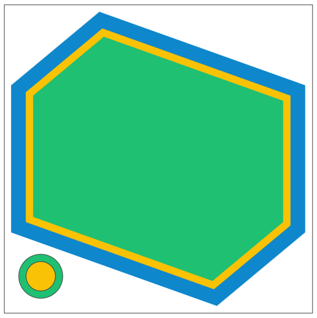
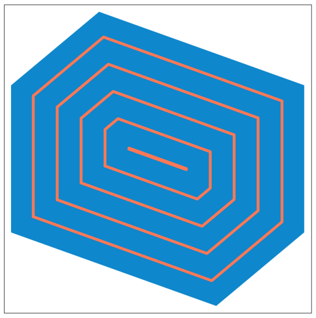
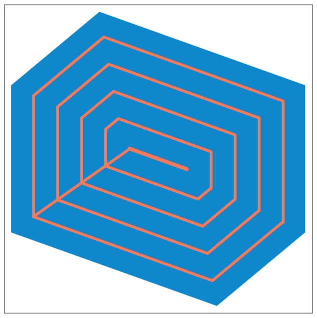
Domain & Vehicle Config.	Constraint Layout	Coverage Path
 Vehicle Radius: 2.5 m Payload Radius: 5.0 m	 Num. of Constraints: 5 Coverage Rate: 100.00%	 Length: 960.96 m Total Turns: 2168.27°
 Vehicle Radius: 5.0 m Payload Radius: 5.0 m	 Num. of Constraints: 5 Coverage Rate: 99.88%	 Length: 928.92 m Total Turns: 2168.27°
 Vehicle Radius: 7.5 m Payload Radius: 5.0 m	 Num. of Constraints: 5 Coverage Rate: 89.28%	 Length: 837.92 m Total Turns: 2168.27°

Table A.2: Spiral Coverage Plans with Changing Vehicle Envelope

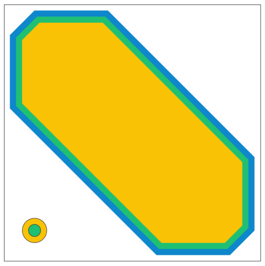
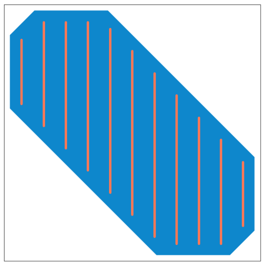
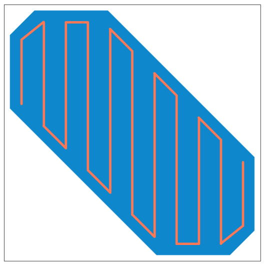
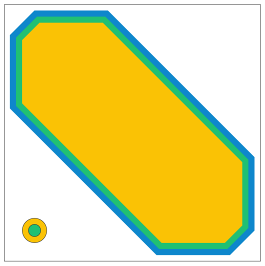
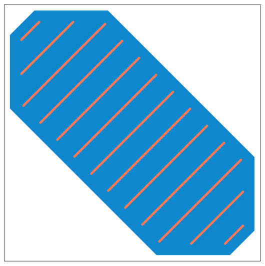
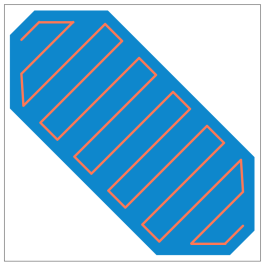
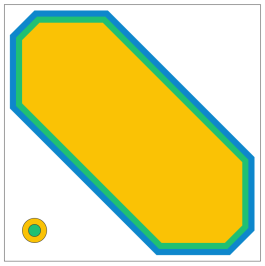
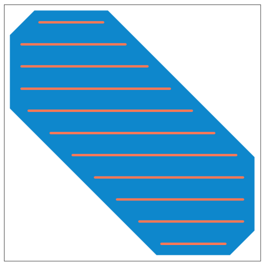
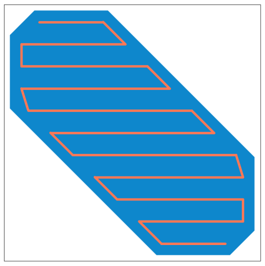
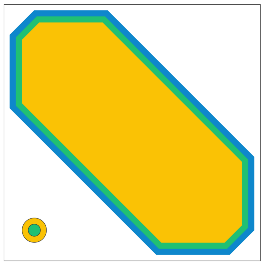
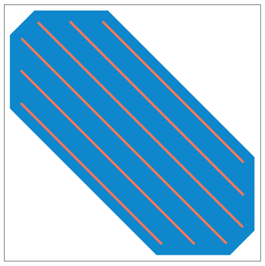
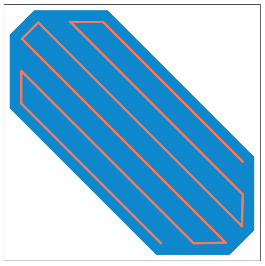
Domain & Vehicle Config.	Constraint Layout	Coverage Path	Path Metrics
			Length: 681.32 m Coverage: 97.37% Total Turns: 1800.00°
			Length: 639.32 m Coverage: 97.33% Total Turns: 2160.00°
			Length: 677.05 m Coverage: 97.10% Total Turns: 1800.00°
			Length: 660.47 m Coverage: 98.50% Total Turns: 900.00°

Table A.3: Boustrophedon Coverage Plans with Changing Transect Orientation

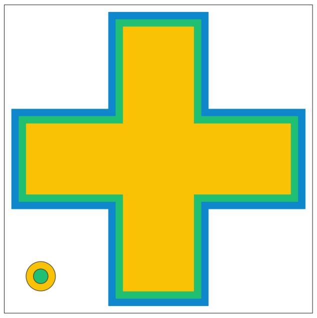
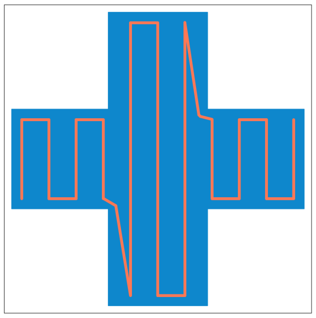
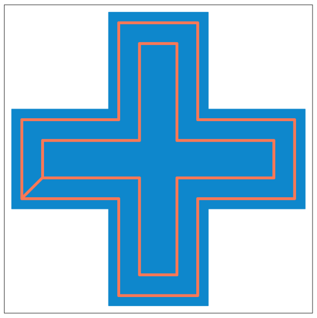
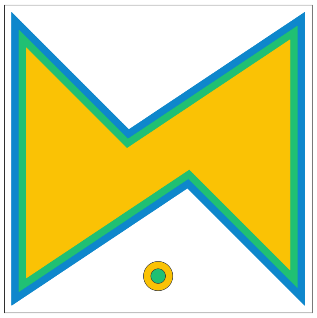
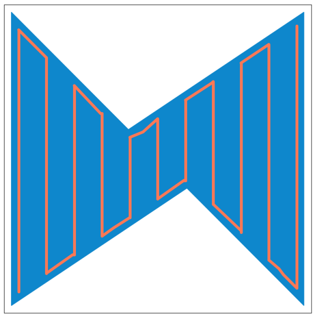
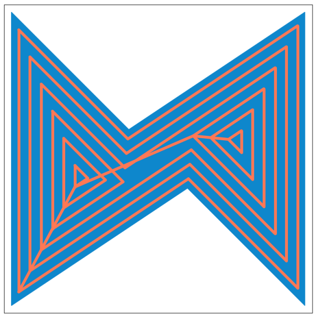
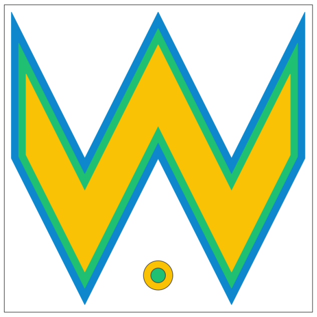
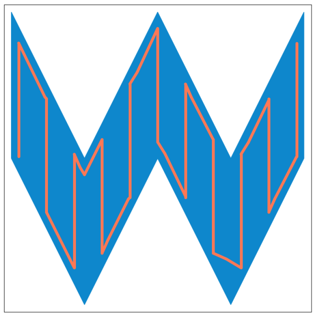
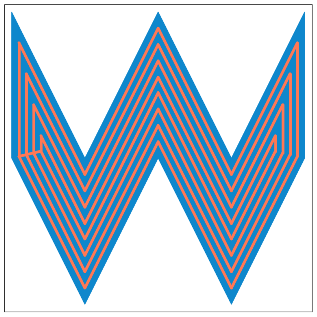
Domain & Vehicle Config.	Boustrophedon Path	Spiral Path
		
Length, Coverage, Turns:	641.13 m, 95.40%, 2035.58°	696.86 m, 92.96%, 2160.00°
		
Length, Coverage, Turns:	729.27 m, 99.17%, 2221.56°	1571.65 m, 99.97%, 4578.21°
		
Length, Coverage, Turns:	629.24 m, 93.53%, 2308.69°	1749.35 m, 99.65%, 4918.62°

Table A.4: Boustrophedon and Spiral Coverage Plans for Different Domains

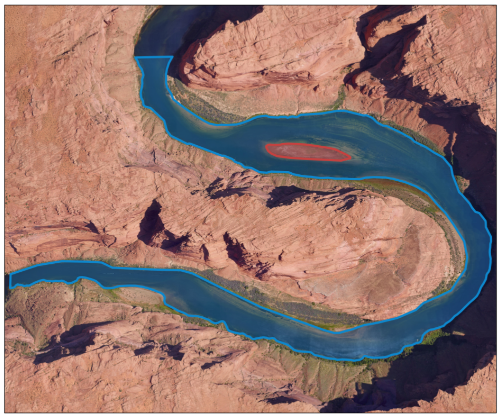
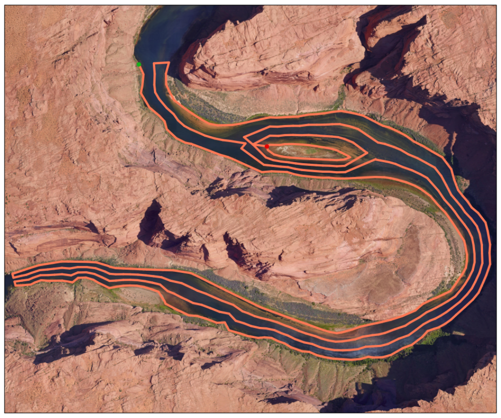
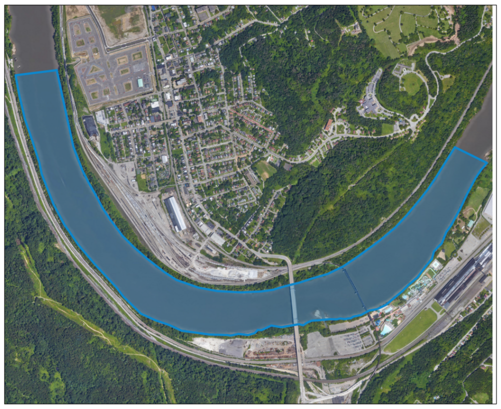
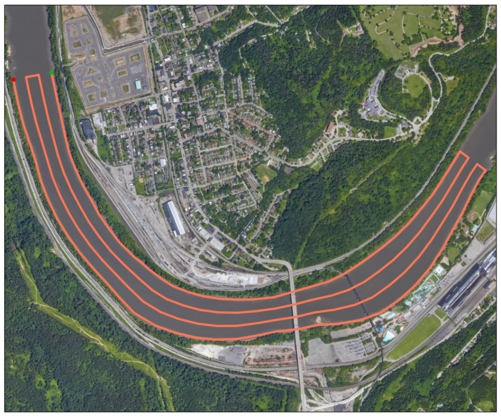
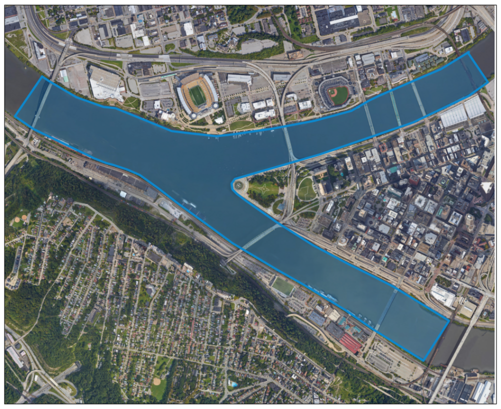
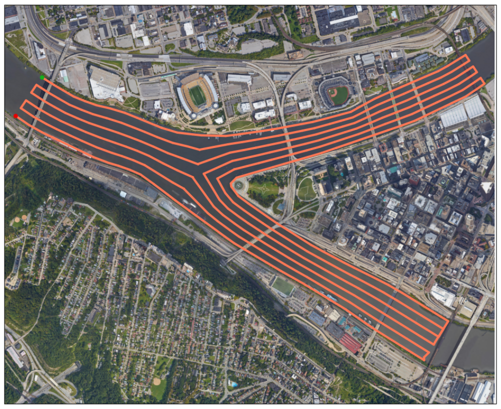
Domain & Target Region	Streamline Boustrophedon Path
	
Colorado River at Horseshoe Bend, AZ	
	
Monongahela River at Pittsburgh, PA	
	
Three Rivers Confluence at Pittsburgh, PA	

Table A.5: Streamline Boustrophedon Coverage Plans for Real World Domains

Appendix B

Full Simulation Results

In this appendix we provide additional figures and complete tables of data from our simulations of various scenarios. The primary reasons for providing this information are to illustrate the impact that our choice of boustrophedon baseline path has on the path execution time, and for completeness since only the best performing boustrophedon path result was presented in the simulation results chapter. It is important to note that since these results are the outputs of computer simulations we tabulate them to as much as seven digits, however, given all simulation assumptions and real-world variability, we do not believe more than three are really meaningful.

B.1 Four Transect Field Experiment Analog

For this experiment, we present the simulation data for all possible boustrophedon baseline paths and directions in table B.1 below.

Vehicle Speed (m/s)	Boustrophedon Path (s)	Rev. Boustrophedon Path (s)	Alt. Boustrophedon Path (s)	Rev. Alt. Boustrophedon Path (s)	Energy-Efficient Sequencing Path (s)
0.375	5107.19	11545.79	11403.86	5152.76	2404.37
0.400	3062.82	4587.92	4528.89	3080.01	1977.19
0.425	2294.73	3100.84	3066.57	2301.52	1709.63
0.450	1878.37	2411.00	2388.82	1879.87	1517.50
0.475	1611.28	2000.82	1985.67	1609.69	1370.84
0.500	1422.54	1724.02	1713.71	1419.67	1251.47
0.525	1280.50	1522.10	1515.75	1276.56	1153.20
0.550	1168.79	1369.10	1364.09	1164.09	1073.01
0.575	1078.10	1247.74	1243.55	1072.52	1004.53
0.600	1002.62	1148.75	1145.06	997.47	945.13
0.625	938.59	1066.26	1062.75	933.79	893.15
0.650	883.40	996.18	992.84	878.91	847.26
0.675	835.23	935.70	932.52	831.02	806.33
0.700	792.72	882.96	879.91	788.73	769.61
0.725	754.86	836.43	833.51	751.08	736.39
0.750	720.88	795.04	792.24	717.28	706.19
0.775	690.17	757.96	755.26	686.73	678.61
0.800	662.26	724.49	721.89	658.98	653.29
0.825	636.74	694.13	691.61	633.59	629.95
0.850	613.31	666.41	663.97	610.30	608.35
0.875	591.73	641.02	638.65	588.82	588.30
0.900	571.72	617.64	615.36	568.91	569.62
0.925	553.14	596.04	593.81	550.45	552.18
0.950	535.84	576.01	573.85	533.23	535.84
0.975	519.66	557.37	555.27	517.14	520.51
1.000	504.53	539.99	537.94	502.09	506.09

Table B.1: Complete Field Experiment Analog Scenario Simulation Results

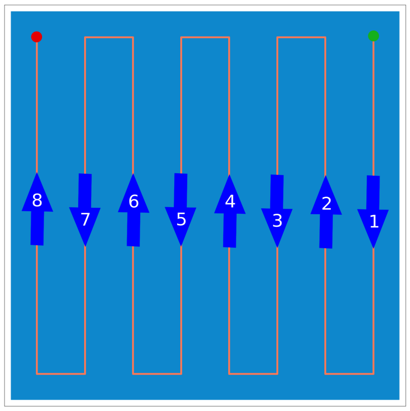
B.2 River Cross-Sections

B.2.1 Wide River Cross-Section

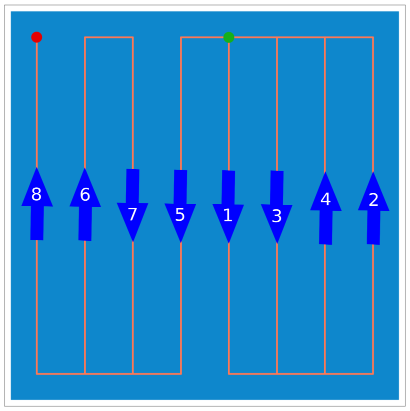
Vehicle Speed (m/s)	Boustro-phedon Path (s)	Reverse Boustrophedon Path (s)	Alternate Boustrophedon Path (s)	Rev. Alt. Boustrophedon Path (s)	Energy-Efficient Path (s)
0.510	5304.47	5304.48	5304.48	5304.47	2141.33
0.525	3504.32	3504.32	3504.32	3504.32	1935.21
0.550	2478.37	2478.37	2478.37	2478.37	1723.93
0.575	2007.25	2007.25	2007.25	2007.25	1573.63
0.600	1721.47	1721.47	1721.47	1721.47	1455.55
0.625	1523.93	1523.93	1523.93	1523.93	1358.37
0.650	1376.64	1376.64	1376.64	1376.64	1276.11
0.675	1261.21	1261.21	1261.21	1261.21	1205.18
0.700	1167.53	1167.53	1167.53	1167.53	1143.07
0.725	1089.49	1089.49	1089.49	1089.49	1088.06
0.750	1023.16	1023.16	1023.16	1023.16	1038.91
0.775	965.86	965.86	965.86	965.86	994.59
0.800	915.74	915.74	915.74	915.74	954.43
0.825	871.38	871.38	871.38	871.38	917.74
0.850	831.77	831.77	831.77	831.77	884.11
0.875	796.15	796.15	796.15	796.15	853.13
0.900	763.86	763.86	763.86	763.86	824.48
0.925	734.44	734.44	734.44	734.44	797.87
0.950	707.52	707.52	707.52	707.52	773.10
0.975	682.68	682.68	682.68	682.68	749.91
1.000	659.79	659.79	659.79	659.79	728.26

Table B.2: Complete Wide River Cross-Section Simulation Results

B.2.2 Square River Cross-Section



(a) Boustrophedon Coverage Path



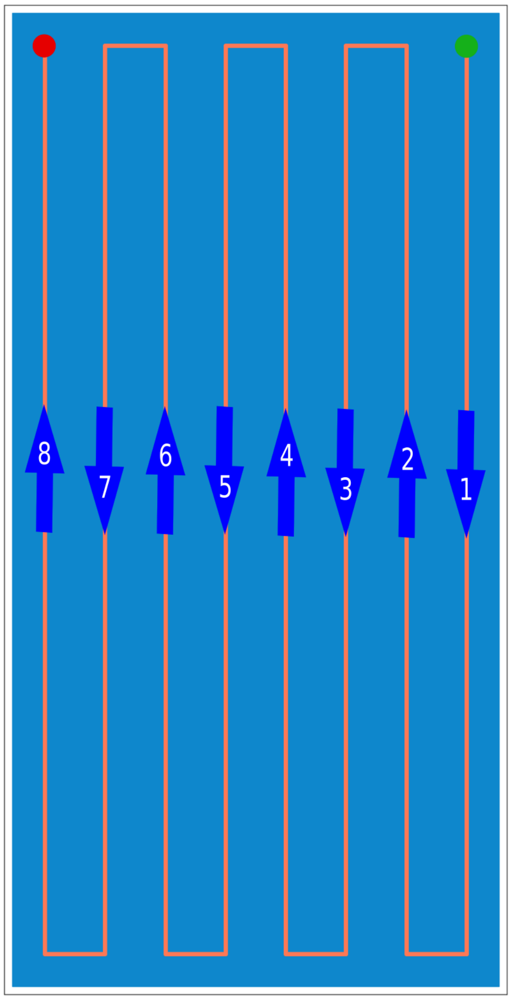
(b) Energy-Efficient Coverage Path

Figure B.1: Baseline and test paths for the square river cross-section scenario

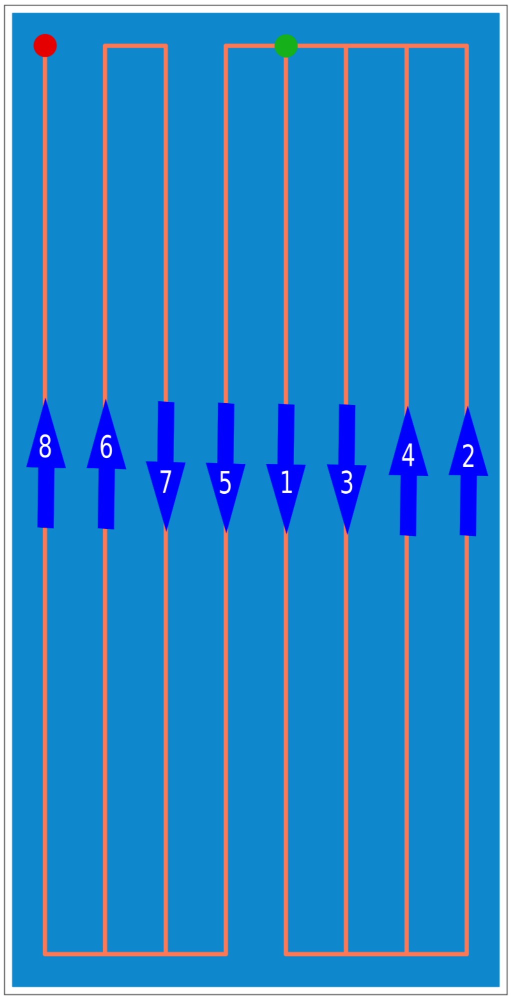
Vehicle Speed (m/s)	Boustrophedon Path (s)	Reverse Boustrophedon Path (s)	Alternate Boustrophedon Path (s)	Rev. Alt. Boustrophedon Path (s)	Energy-Efficient Path (s)
0.510	11765.01	11765.01	11765.01	11765.01	3702.30
0.525	7651.97	7651.97	7651.97	7651.97	3411.20
0.550	5334.73	5334.73	5334.73	5334.73	3080.45
0.575	4283.97	4283.97	4283.97	4283.97	2831.55
0.600	3652.48	3652.48	3652.48	3652.48	2630.32
0.625	3219.25	3219.25	3219.25	3219.25	2461.88
0.650	2898.22	2898.22	2898.22	2898.22	2317.79
0.675	2647.95	2647.95	2647.95	2647.95	2192.50
0.700	2445.75	2445.75	2445.75	2445.75	2082.18
0.725	2277.94	2277.94	2277.94	2277.94	1984.03
0.750	2135.81	2135.81	2135.81	2135.81	1896.01
0.775	2013.42	2013.42	2013.42	2013.42	1816.49
0.800	1906.61	1906.61	1906.61	1906.61	1744.17
0.825	1812.34	1812.34	1812.34	1812.34	1678.08
0.850	1728.37	1728.37	1728.37	1728.37	1617.39
0.875	1652.95	1652.95	1652.95	1652.95	1561.41
0.900	1584.73	1584.73	1584.73	1584.73	1509.49
0.925	1522.69	1522.69	1522.69	1522.69	1461.35
0.950	1465.94	1465.94	1465.94	1465.94	1416.43
0.975	1413.77	1413.77	1413.77	1413.77	1374.46
1.000	1365.63	1365.63	1365.63	1365.63	1335.10

Table B.3: Complete Square River Cross-Section Simulation Results

B.2.3 Long River Cross-Section



(a) Boustrophedon Coverage Path



(b) Energy-Efficient Coverage Path

Figure B.2: Baseline and test paths for the long river cross-section scenario

Vehicle Speed (m/s)	Boustrophedon Path (s)	Reverse Boustrophedon Path (s)	Alternate Boustrophedon Path (s)	Rev. Alt. Boustrophedon Path (s)	Energy-Efficient Path (s)
0.510	24686.05	24686.07	24686.07	24686.05	6824.16
0.525	15947.30	15947.30	15947.30	15947.30	6363.24
0.550	11047.52	11047.51	11047.51	11047.52	5793.55
0.575	8837.38	8837.38	8837.38	8837.38	5347.32
0.600	7514.48	7514.48	7514.48	7514.48	4979.86
0.625	6609.88	6609.88	6609.88	6609.88	4668.97
0.650	5941.35	5941.36	5941.36	5941.35	4401.11
0.675	5421.40	5421.40	5421.40	5421.40	4167.09
0.700	5002.16	5002.16	5002.16	5002.16	3960.34
0.725	4654.86	4654.86	4654.86	4654.86	3775.95
0.750	4361.12	4361.12	4361.12	4361.12	3610.22
0.775	4108.54	4108.54	4108.54	4108.54	3460.25
0.800	3888.36	3888.36	3888.36	3888.36	3323.72
0.825	3694.28	3694.28	3694.28	3694.28	3198.79
0.850	3521.54	3521.54	3521.54	3521.54	3083.94
0.875	3366.56	3366.57	3366.57	3366.56	2977.92
0.900	3226.51	3226.51	3226.51	3226.51	2879.68
0.925	3099.19	3099.19	3099.19	3099.19	2788.32
0.950	2982.82	2982.83	2982.83	2982.82	2703.14
0.975	2875.93	2875.93	2875.93	2875.93	2623.49
1.000	2777.33	2777.33	2777.33	2777.33	2548.77

Table B.4: Coverage Path Execution Time at Various Vehicle Speeds

B.3 River Bend Scenario

Speed (m/s)	Stream (s)	Rev. Stream (s)	Biased Stream (s)	Rev. Biased Stream (s)	EE (s)	Biased EE (s)	Bous. (s)	Rev. Bous. (s)
0.510	46031.46	47332.68	49778.00	30973.45	21123.63	10821.05	26015.83	25081.17
0.525	29000.69	28920.82	27142.81	22729.91	16783.21	9906.76	17628.18	16789.95
0.550	19900.7	19594.25	17558	16888.24	13437.53	8937.75	13006.9	12270.24
0.575	15802.07	15525.94	13792.83	13854.53	11555.16	8229	10820.63	10161.68
0.600	13355.14	13130.99	11674.68	11921.9	10279	7663.63	9453.85	8858.82
0.625	11689.47	11510.67	10271.81	10557.03	9331.3	7194.04	8484.55	7943.72
0.650	10464.64	10322.67	9252.47	9529.27	8588.33	6793.69	7745.69	7251.41
0.675	9517.01	9403.58	8466.23	8720.78	7983.81	6445.99	7155.68	6701.84
0.700	8756.38	8665.79	7834.67	8063.61	7478.56	6139.74	6668.95	6250.57
0.725	8129.31	8056.58	7311.96	7516.3	7047.59	5866.93	6257.72	5870.63
0.750	7601.15	7542.47	6869.47	7051.62	6673.96	5621.9	5903.84	5544.51
0.775	7148.51	7101.25	6488.21	6651	6345.87	5400.06	5594.86	5260.34
0.800	6755.42	6717.25	6154.97	6300.74	6054.67	5197.61	5321.83	5009.62
0.825	6409.97	6379.32	5860.58	5991.45	5794	5012.46	5078.31	4786.1
0.850	6103.47	6078.76	5597.71	5715.53	5558.52	4841.54	4859.28	4585.15
0.875	5829.15	5809.3	5361.08	5467.87	5344.84	4683.73	4660.86	4403.22
0.900	5581.81	5566.03	5146.76	5243.61	5149.24	4536.89	4480.06	4237.38
0.925	5357.29	5345.03	4951.21	5039.51	4969.59	4400.15	4314.38	4085.4
0.950	5152.56	5143.09	4771.99	4852.76	4804.02	4272.25	4161.88	3945.45
0.975	4964.88	4957.87	4607.06	4681.24	4650.68	4152.5	4020.95	3816.02
1.000	4792.12	4786.91	4454.45	4522.64	4507.91	4040.17	3890.19	3695.87

Table B.5: River Bend Coverage Path Execution Time at Various Vehicle Speeds

Appendix C

Full Field Results

In this appendix, we present our complete field result data as it was recorded during the experiments. Since battery changes did not occur after every run, the starting and ending power and current consumption on the integrated meter aboard the vehicle were recorded. This data was then used to compute the new power and current usage for that run.

C.1 North Carolina Cape Fear River 3 Transect Scenario

Coverage Path	Starting Power (Wh)	Starting Current (Ah)	Ending Power (Wh)	Ending Current (Ah)	Net Power Usage (Wh)	Net Current Usage (Ah)	Runtime
Boustrophedon	8.9	0.741	15.5	1.285	6.6	0.544	14:57.75
Energy-Efficient	15.7	1.301	21.4	1.773	5.7	0.472	12:49.35
Boustrophedon	21.5	1.781	27.9	2.309	6.4	0.528	14:39.48
Energy-Efficient	28.0	2.316	33.9	2.808	5.9	0.492	13:19.35
Boustrophedon	34.0	2.814	40.3	3.335	6.3	0.521	14:19.84
Energy-Efficient	43.4	3.593	48.9	4.050	5.5	0.457	12:26.00
Boustrophedon	49.0	4.056	55.1	4.563	6.1	0.507	13:47.32
Energy-Efficient	0.5	0.048	6.3	0.525	5.8	0.477	12:53.32
Boustrophedon	6.4	0.536	12.8	1.064	6.4	0.528	14:04.99
Energy-Efficient	12.9	1.070	18.5	1.535	5.6	0.465	12:31.05
Boustrophedon	18.6	1.543	25.3	2.096	6.7	0.553	14:51.57
Energy-Efficient	25.4	2.102	30.5	2.531	5.1	0.429	11:36.51
Boustrophedon	30.6	2.539	36.3	3.005	5.7	0.466	12:44.67
Energy-Efficient	39.8	3.296	45.1	3.736	5.3	0.440	11:48.42
Boustrophedon	45.2	3.748	51.1	4.228	5.9	0.480	12:50.94
Energy-Efficient	72.6	6.027	77.9	6.469	5.3	0.442	11:43.38

Table C.1: NC Cape Fear River 3 Transect Coverage Path Experimental Results

C.2 North Carolina Cape Fear River 4 Transect Scenario

Coverage Path	Starting Power (Wh)	Starting Current (Ah)	Ending Power (Wh)	Ending Current (Ah)	Net Power Usage (Wh)	Net Current Usage (Ah)	Runtime
Boustrophedon	14.1	1.175	21.7	1.803	7.6	0.628	18:17.81
Energy-Efficient	21.8	1.809	28.1	2.335	6.3	0.526	15:04.58
Boustrophedon	28.2	2.34	35.8	2.968	7.6	0.628	17:57.74
Energy-Efficient	35.9	2.973	42.8	3.55	6.9	0.577	16:32.40
Boustrophedon	42.9	3.557	50.7	4.201	7.8	0.644	18:42.03
Energy-Efficient	0.5	0.046	7.5	0.622	7	0.576	16:39.82
Boustrophedon	7.6	0.628	15.7	1.301	8.1	0.673	19:40.99
Energy-Efficient	15.8	1.307	22.9	1.895	7.1	0.588	17:10.64
Boustrophedon	23	1.901	31	2.56	8	0.659	19:15.85
Energy-Efficient	31.1	2.566	37.9	3.125	6.8	0.559	16:20.07
Boustrophedon	37.9	3.129	45.4	3.746	7.5	0.617	18:00.38
Energy-Efficient	45.5	3.756	52.2	4.309	6.7	0.553	16:14.88
Boustrophedon	0.4	0.037	8.8	0.727	8.4	0.69	19:49.74
Energy-Efficient	8.8	0.732	15.7	1.301	6.9	0.569	16:45.71
Boustrophedon	15.8	1.311	23.6	1.949	7.8	0.638	18:34.02
Energy-Efficient	23.7	1.957	30.2	2.497	6.5	0.54	15:40.93
Boustrophedon	30.3	2.503	37.9	3.13	7.6	0.627	18:10.21
Energy-Efficient	38	3.136	44.6	3.683	6.6	0.547	15:42.48
Boustrophedon	0.4	0.035	7.7	0.637	7.3	0.602	17:10.25
Energy-Efficient	7.8	0.644	14	1.165	6.2	0.521	14:52.32

Table C.2: NC Cape Fear River 4 Transect Coverage Path Experimental Results